Aspects algorithmiques de la comparaison d'éléments biologiques

Florian Sikora

UNIVERSITÉ PARIS-EST

THÈSE

pour obtenir le grade de
DOCTEUR DE L'UNIVERSITÉ PARIS-EST

Aspects algorithmiques de la comparaison d'éléments biologiques

Sous la direction de Stéphane VIALETTE *et l'encadrement de* Guillaume BLIN

Spécialité **informatique**
École doctorale MSTIC
Soutenue publiquement par **Florian Sikora**
le **30 septembre 2011**

JURY :

Guillaume BLIN,	Paris-Est,	examinateur,
Maxime CROCHEMORE,	Paris-Est & King's College London,	examinateur,
Alain DENISE,	Paris-Sud,	rapporteur,
Vincent LACROIX,	Claude Bernard Lyon 1,	examinateur,
Ioan TODINCA,	Orléans,	rapporteur,
Stéphane VIALETTE,	Paris-Est,	directeur.

Table des matières

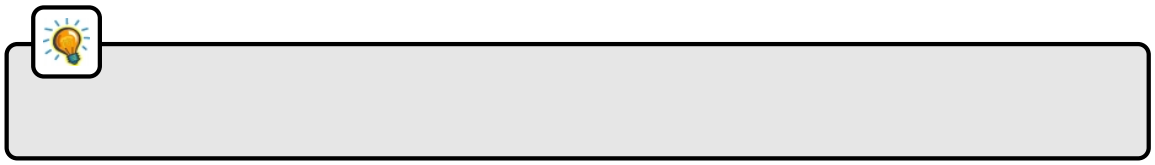
Notes de lecture

Les résultats connus dans la littérature sont présentés en tant que *théorèmes*, tandis que les résultats obtenus durant cette thèse sont présentés en tant que *propositions*.

Pour des raisons de lisibilité et de bibliographie, les noms des problèmes algorithmiques étudiés et utilisés sont laissés en anglais – un index répertoriant leur définition est disponible en fin de document.

Les illustrations dont la source n'est pas précisée sont l'œuvre de l'auteur du manuscrit. La majorité de ces dernières ont été effectuées à l'aide de tikz [1].

Le lecteur trouvera régulièrement au fil des pages des encarts de ce type :



Dans la Partie I, ils sont essentiellement utilisés pour donner des informations supplémentaires, non indispensables à la suite, voire des anecdotes ou des points historiques. Dans les deux parties suivantes, ces encarts mettent en avant des pistes pour des recherches futures, des idées non abouties ou encore des problèmes ouverts sur lesquels je n'ai pas pu obtenir de réponse.

1. http://sourceforge.net/projects/pgf/

Publications issues de cette thèse

L'ordre des auteurs est alphabétique, sauf pour [YSB+11].

Journaux

[BRSV11b] Guillaume BLIN, Romeo RIZZI, Florian SIKORA et Stéphane VIALETTE : Minimum Mosaic Inference of a Set of Recombinants. *International Journal of Foundations of Computer Science (IJFCS)*, 2011. Accepté pour publication.

[GS11] Sylvain GUILLEMOT et Florian SIKORA : Finding and counting vertex-colored subtrees. *Algorithmica*, 2011. Accepté pour publication.

[BSV10b] Guillaume BLIN, Florian SIKORA et Stéphane VIALETTE : Querying Graphs in Protein-Protein Interactions Networks using Feedback Vertex Set. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 7(4) :628–635, 2010. Special Issue-ISBRA 2009-Bioinformatics Research and Applications.

Actes de conférences

[BBD+12] Guillaume BLIN, Paola BONIZZONI, Riccardo DONDI, Romeo RIZZI et Florian SIKORA : Complexity Insights of the Minimum Duplication Problem. *In Proceedings of the 38th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'12)*, *Lecture Notes in Computer Science*. Springer, 2012. À paraître.

[BFMB+11] Guillaume BLIN, Guillaume FERTIN, Hafedh MOHAMED-BABOU, Irena RUSU, Florian SIKORA et Stéphane VIALETTE : Algorithmic Aspects of Heterogeneous Biological Networks Comparison. *In* Weifan WANG, Xuding ZHU et Ding-Zhu DU, éditeurs : *Proceedings of the 5th Annual International Conference on Combinatorial Optimization and Applications (COCOA'11)*, volume 6831 de *Lecture Notes in Computer Science*, pages 272–286. Springer, 2011.

[BFSV09] Guillaume BLIN, Guillaume FERTIN, Florian SIKORA et Stéphane VIALETTE : The Exemplar Breakpoint Distance for Non-trivial Genomes Cannot Be Approximated. *In* Sandip DAS et Ryuhei UEHARA, éditeurs : *Proceedings of the 3rd International Workshop Algorithms and Computation (WALCOM)*, volume 5431 de *Lecture Notes in Computer Science*, pages 357–368. Springer, 2009.

[BRSV11a] Guillaume BLIN, Romeo RIZZI, Florian SIKORA et Stéphane VIALETTE : Minimum Mosaic Inference of a Set of Recombinants. *In* Alex POTANIN et Taso VIGLAS, éditeurs : *Proceedings of the 17th Computing : the Australasian Theory Symposium (CATS)*, volume 119 de *CRPIT*, pages 23–30. ACS, 2011.

[BSV09] Guillaume BLIN, Florian SIKORA et Stéphane VIALETTE : Querying Protein- Protein Interaction Networks. *In* Ion I. MANDOIU, Giri NARASIMHAN et Yanqing ZHANG, éditeurs : *Proceedings of the 5th International Symposium Bioinformatics Research and Applications (ISBRA)*, volume 5542 de *Lecture Notes in Bioinformatics*, pages 52–62. Springer-Verlag, 2009.

[BSV10a] Guillaume BLIN, Florian SIKORA et Stéphane VIALETTE : GraMoFoNe : a Cytoscape plugin for querying motifs without topology in Protein-Protein Interactions networks. *In* Hisham AL-MUBAID, éditeur : *Proceedings of the 2nd International Conference on Bioinformatics and Computational Biology (BICoB)*, pages 38–43. International Society for Computers and their Applications (ISCA), 2010.

[GS10] Sylvain GUILLEMOT et Florian SIKORA : Finding and counting vertex-colored subtrees. In Petr HLINEÝ et Antonín KUCERA, éditeurs : *Proceedings of the 35th International Symposium on Mathematical Foundations of Computer Science (MFCS'10)*, volume 6281 de *Lecture Notes in Computer Science*, pages 405–416, Brno, Czech Republic, August 2010. Springer.

En cours de soumission

[YSB+11] Xiao YANG, Florian SIKORA, Guillaume BLIN, Sylvie HAMEL, Romeo RIZZI et Srinivas ALURU : An Algorithmic View on Multi-related-segments : a new unifying model for approximate common interval.

Introduction

Contrairement aux prédictions effectuées il y a quelques années, le nombre de gènes chez l'homme ne serait que de 25.000 [Cla05]. Si, par exemple, ce nombre est comparé aux 45.000 gènes du riz, il est légitime de supposer que la relative complexité d'un organisme n'est pas directement liée à son nombre de gènes, tout du moins pas de façon linéaire. En effet, lister les gènes présents dans un organisme est parfois discutable, car il ne suffit pas de relever la présence d'un gène dans une cellule pour que le caractère correspondant soit automatiquement exprimé.

L'une des méthodes utilisées pour mieux comprendre les liens complexes entre le génotype (ensemble des gènes relevés chez un individu) et le phénotype (ensemble des caractères observés chez un individu) consiste à étudier les relations entre différents éléments biologiques (entre les protéines, entre les métabolites...). Celles-ci forment ce qui est appelé un *réseau biologique*. Leur utilisation permet la réponse à de nouvelles problématiques. Les réseaux principalement utilisés dans cette thèse sont les *réseaux d'interactions entre protéines* ainsi que les *réseaux métaboliques*.

De manière algorithmique, nous représentons un réseau biologique par un graphe, où les sommets représentent les éléments biologiques, et où les arêtes représentent les interactions entre les éléments correspondants. Ce graphe est, de plus, coloré sur les sommets. La couleur représente alors une certaine propriété biologique de l'élément (une classe de protéines, une classe de réactions...). Un problème largement étudié dans la littérature consiste à déterminer si une certaine requête, représentée par un chemin, un arbre ou un graphe, est présente dans le graphe représentant le réseau biologique. Ce problème, analogue à celui de la recherche de motif dans un texte ou de requête dans une base de données, est fondamental en biologie. Il permet en effet d'identifier les parties importantes d'un grand réseau ou encore de découvrir des fonctions communes entre différentes espèces. Cependant, ce type de problème est rapidement difficile algorithmiquement puisqu'il est proche des problèmes de morphismes de graphes.

Récemment, Lacroix, Fernandes et Sagot ont introduit une variante du problème de requête dans un réseau biologique, avec un problème appelé GRAPH MOTIF [LFS06]. Il a été montré que la topologie des requêtes était parfois inconnue ou tout simplement non pertinente. Par conséquent, dans ce nouveau formalisme, la requête (ou le motif) est

seulement un multi-ensemble de couleurs. Le problème consiste alors à déterminer s'il existe un sous-graphe connexe, contenant toutes les couleurs du motif (voir Figure 1). L'approche choisie est plus *fonctionnelle* que *topologique*.

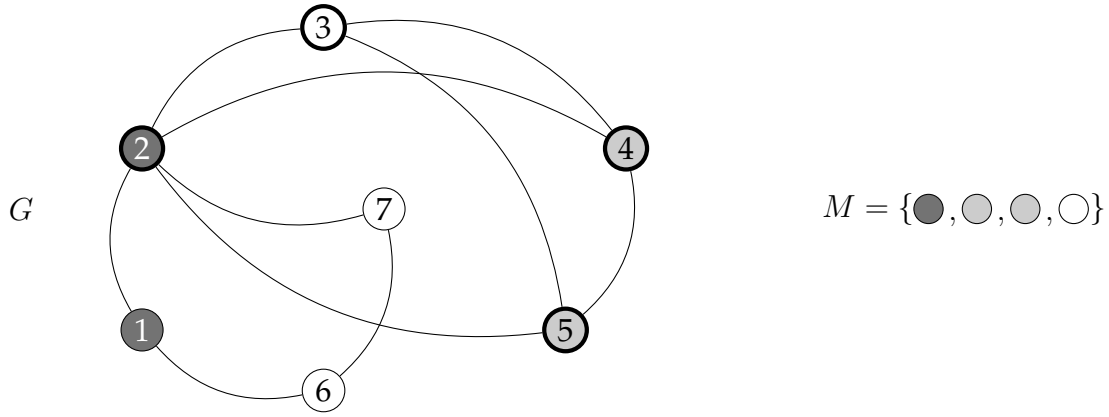


FIGURE 1 – Un graphe G coloré sur les sommets et un motif M, multi-ensemble de taille 4 utilisant trois couleurs. Une solution au problème est donnée en gras et contient les sommets $\{2, 3, 4, 5\}$. Le sous-graphe induit par cette solution est connexe, et contient toutes les couleurs demandées dans M.

Ce manuscrit contient trois parties distinctes. L'objet de la Partie I est le rappel de bases d'algorithmique, de techniques utiles pour contourner la difficulté d'un problème (Chapitre 1), mais aussi le rappel des notions centrales d'ADN, d'ARN, de gènes et de protéines, ainsi que la présentation plus en détail des réseaux biologiques (Chapitre 2), au centre des problématiques de la partie suivante.

La Partie II de cette thèse porte sur une étude algorithmique des problèmes concernant les réseaux biologiques (Chapitres 3 et 7), et plus particulièrement celle du problème GRAPH MOTIF (Chapitres 4 et 5). Le problème ayant été montré difficile algorithmiquement (NP-*Complet*) même sous de fortes contraintes, il convient de proposer des alternatives. C'est pourquoi, j'ai proposé avec mes co-auteurs des algorithmes efficaces de *complexité paramétrée* (algorithmes ayant une complexité exponentielle selon un paramètre présupposé petit et non selon la taille totale de l'instance), ainsi que des règles réduisant la taille des instances considérées. J'ai également étudié dans cette thèse différentes variantes du problème, autorisant plus de souplesse, où des *algorithmes d'approximations* (algorithmes ne retournant pas la solution optimale mais dont le ratio d'erreur est borné) sont envisageables. Malheureusement, nous verrons qu'il n'existe pas d'algorithme d'approximation avec un faible ratio, même pour des instances très contraintes. Enfin, je me suis intéressé à la version de comptage du problème, ainsi qu'à une alternative sur la contrainte de connexité, en demandant que le sous-graphe soit un module qui respecte les couleurs du motif. Certains de ces résultats ne sont pas encore publiés, d'autres l'ont été dans [BSV09, BSV10b, GS10, BFMB+11].

Je pense que la bio-informatique doit, par nature, également fournir l'implémentation d'algorithmes et proposer des solutions utilisables en pratique par les biologistes.

C'est pourquoi, j'ai proposé une implémentation d'un algorithme répondant au problème de recherche d'une requête topologique (Chapitre 3) ainsi qu'un greffon pour le logiciel Cytoscape (Chapitre 6 et publié dans [BSV10a]). Cytoscape est un logiciel libre bien connu des biologistes et largement utilisé. Plutôt que de proposer un programme indépendant, nécessitant une installation et ayant peu de visibilité, il m'a semblé pertinent de participer à un projet libre ayant une large visibilité dans la communauté. Il est possible d'obtenir ce greffon directement depuis l'interface de Cytoscape ; il a d'ailleurs été téléchargé plus de 350 fois à ce jour.

Au gré d'opportunités et de collaborations scientifiques, j'ai été amené à travailler sur des thèmes non directement liés aux réseaux biologiques. Ceci donnant lieu à un ensemble de résultats annexes présentés brièvement Partie III. Dans cette partie, les objets combinatoires étudiés sont plus divers que dans les parties précédentes. En effet, j'ai utilisé, outre les graphes, des chaînes de caractères, des permutations ou encore des arbres. Cependant, la démarche scientifique adoptée reste la même : depuis une formalisation d'un problème biologique, déterminer ses instances difficiles algorithmiquement et proposer des solutions pour contourner cette difficulté (ou prouver que de telles solutions sont impossibles à trouver sous des hypothèses fortes). Une fois données les motivations et les résultats connus, j'ai choisi de seulement énumérer les contributions, sans en donner les preuves. Celles-ci sont disponibles dans les publications correspondantes : [BRSV11a, BFSV09, YSB+11, BBD+12].

Première partie

Quelques bases...

Chapitre 1

Généralités algorithmiques

Contenu

Le but de ce chapitre est de présenter les bases algorithmiques nécessaires à la compréhension de la suite de cette thèse. Nous présenterons, dans un premier temps, quelques éléments de la théorie des graphes, avant d'aborder la théorie de la complexité classique. Le chapitre est conclu par un survol de quelques solutions algorithmiques pour contourner la difficulté d'un problème. Nous n'entrerons pas dans les détails. Ce chapitre ne se substitue donc en aucun cas aux ouvrages de références cités tout au long de celui-ci.

1.1 Graphes

Nous introduisons ici seulement quelques bases sur les graphes. D'autres définitions importantes seront données tout au long de ce document, lorsqu'elles seront nécessaires. Le lecteur désirant plus de détails pourra se référer par exemple au livre de Reinhard Diestel [Die05].

Le *graphe* est un objet abondamment utilisé en pratique, dans des domaines divers et variés. La théorie sous-jacente à leur étude, la *théorie des graphes*, est également largement étudiée. La terminologie utilisée est agréablement intuitive. Ainsi, la plupart des termes employés sont explicites et se retiennent assez facilement.

Un graphe $G = (V, E)$ est constitué d'un ensemble V (de l'anglais *vertices*) de *sommets* (ou *nœuds*) ainsi que d'un ensemble $E \subseteq V \times V$ (de l'anglais *edges*) de paires de sommets,

nommées *arêtes*. Dans la suite, les graphes sont considérés sans boucle, c'est-à-dire sans arête de la forme $\{u, u\}$.

Des graphes se trouvent dans de nombreuses applications. Un graphe peut représenter le réseau Internet, des réseaux biologiques, le réseau des connaissances entre citoyens, des réseaux de communications, des planifications de tâches... Nous avons choisi d'illustrer les notions de cette section à l'aide du réseau du métro de Paris [1]. Nous pouvons en effet représenter formellement ce réseau à l'aide d'un graphe, où chaque station est un sommet, et où une voie ferrée reliant deux stations est représentée par une arête.

En toute généralité, les voies ferrées entre les stations permettent une circulation dans les deux sens. Le graphe alors associé est donc dit *non-orienté*, car il contient des arêtes. Une arête entre deux sommets u et v est notée comme un ensemble à deux éléments $\{u, v\}$. Le graphe représentant le réseau du métro (Figure 1.1) est non-orienté.

Nous pouvons cependant désirer donner une orientation à ces liens entre deux sommets. Par exemple, les trains ne circulent que dans un seul sens entre les stations `Pré St-Gervais` et `Botzaris` sur la ligne 7bis (voir Figure 1.2). Nous parlons alors d'*arc* plutôt que d'arête. L'ensemble des arcs d'un graphe est noté A (de l'anglais Arcs) et est également constitué d'un ensemble de paires de sommets, c'est-à-dire $A \subseteq V \times V$. Un arc entre deux sommets u et v est noté (u, v) si l'orientation est de u vers v, et (v, u) sinon. Un graphe $G = (V, A)$ contenant uniquement des arcs est dit *orienté*. Nous manipulerons dans la suite de cette thèse essentiellement des graphes non-orientés. Ainsi, il faudra considérer les graphes comme tels lorsque aucune précision ne sera donnée.

Dans la majorité des cas, E et A sont des ensembles simples, c'est-à-dire sans répétition. Dans le cas contraire, ce sont des multi-ensembles, où plusieurs liens entre deux sommets sont possibles et de la notion de *multi-graphe* associée. C'est par exemple le cas dans le graphe du métro Figure 1.1 où l'on trouve deux arêtes reliant les stations `Gare du Nord` et `Gare de l'Est`.

La taille d'un graphe, lorsque aucune précision ne sera donnée, fera référence à son nombre de sommets, c'est-à-dire à $|V|$, souvent noté n.

Les graphes peuvent être *pondérés*. Une fonction de poids est alors ajoutée aux arêtes (ou arcs), y associant un nombre, souvent réel, c'est-à-dire $G = (V, E, w)$, où w est une fonction telle que $w : E \to \mathbb{R}$. Autrement dit, pour chaque arête $e \in E$, la valeur $w(e)$ donne le *poids* de l'arête. Le graphe du métro Figure 1.1 peut être pondéré par la distance entre les stations.

Deux sommets u et v de V sont dits *voisins* (ou *adjacents*) si $e = \{u, v\} \in E$, les sommets u et v sont alors *incidents* à l'arête e. Au contraire, u et v sont dits *indépendants* si $\{u, v\} \notin E$. Dans le graphe du métro Figure 1.1, les stations `Père Lachaise` et `Gambetta` sont voisines, en revanche, les stations `Gambetta` et `Gare de l'Est` sont indépendantes. Ces notions sont identiques dans un graphe orienté, en remplaçant les

1. `http://ratp.fr/`

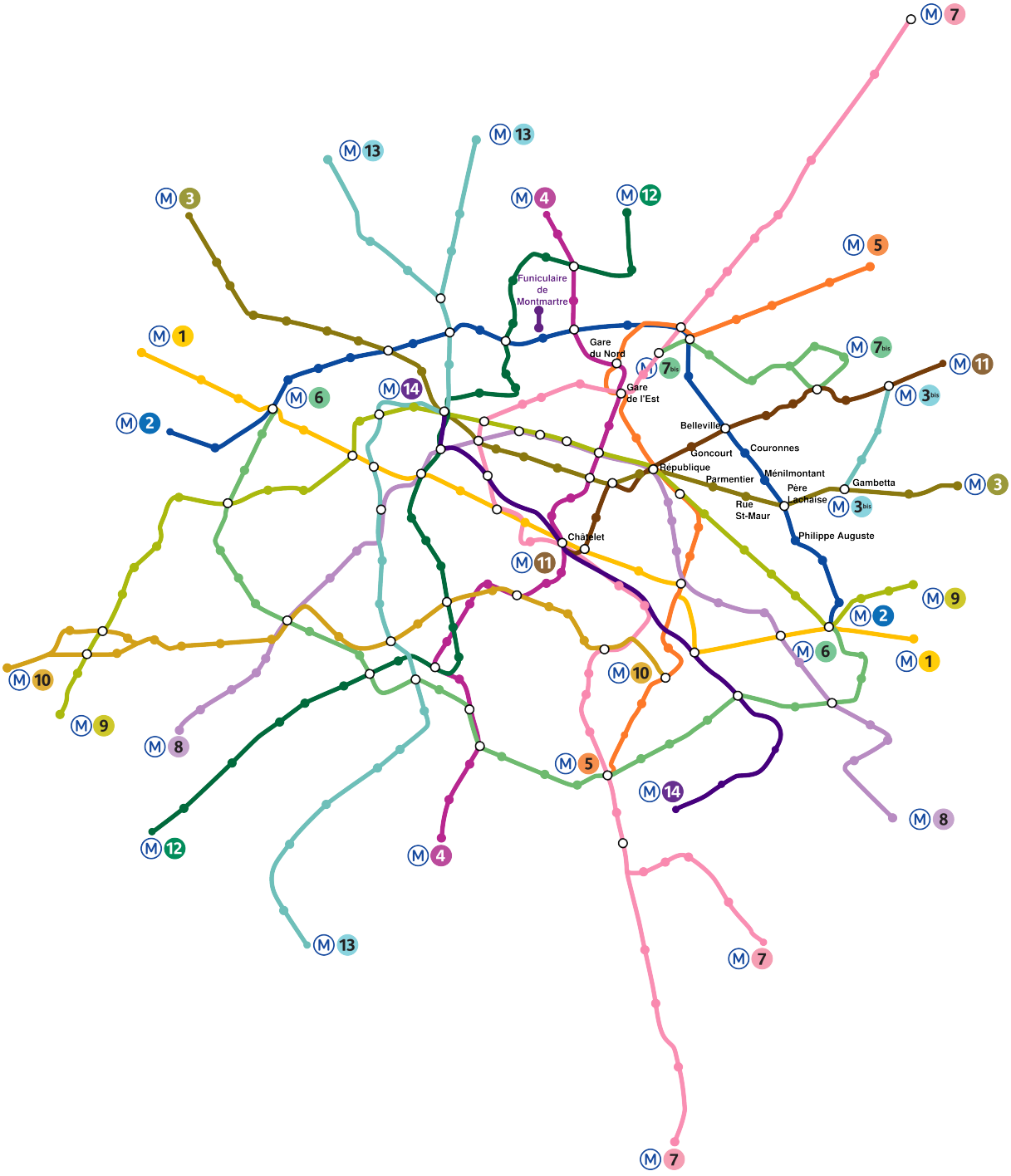


FIGURE 1.1 – Graphe représentant le métro de Paris. Pour des raisons de lisibilité, les extrémités de certaines lignes ainsi que le nom de la plupart des stations ont été omis. Source Wikipedia.

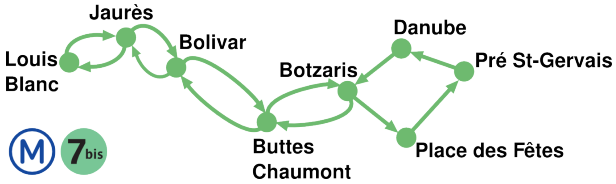


FIGURE 1.2 – Graphe orienté représentant la ligne 7bis du métro de Paris.

arcs orientés par une arête non orientée. Ainsi, deux sommets sont voisins quel que soit le sens de l'arc les reliant.

L'ensemble des voisins d'un sommet $u \in V$, noté $N(u)$, est constitué des sommets partageant une arête avec u :

$$N(u) = \{v : \{u, v\} \in E\}.$$

Le nombre de ces voisins est appelé *degré* de u, noté $d(u)$:

$$d(u) = |N(u)|.$$

Dans le graphe du métro, $N($`Père Lachaise`$) = \{$`Ménilmontant`, `Gambetta`, `Philippe Auguste`, `Rue St-Maur`$\}$ et $d($`Père Lachaise`$) = 4$. Un sommet ayant un degré nul est dit *isolé*. La notion de degré se généralise au graphe, où le degré d'un graphe $G = (V, E)$, noté $\Delta(G)$, est égal au degré le plus élevé de ses sommets :

$$\Delta(G) = \max_{v \in V} d(v).$$

Le degré du graphe du métro Figure 1.1 est égal à 9 car le degré du sommet `Châtelet` est égal à 9 et aucun autre sommet n'a un degré strictement supérieur.

Un sous-ensemble de sommets $V' \subseteq V$ d'un graphe donne un *sous-graphe induit* de G, noté $G[V']$ ou $G' = (V', E')$, où E' contient les arêtes de E dont les deux sommets incidents sont dans V'. Plus formellement, $E' = \{\{u, v\} : \{u, v\} \in E, u \in V', v \in V'\}$. Le sous-graphe induit par l'ensemble des stations traversées par la ligne 7 du graphe du métro se trouve Figure 1.3.

Un *chemin* est une séquence de sommets telle que chaque sommet possède une arête avec le sommet suivant de la séquence. Plus formellement, un chemin à n sommets est noté $P_n = (u_1, u_2, \ldots, u_n)$, où l'arête $\{u_i, u_{i+1}\} \in E$ pour $1 \leq i \leq n - 1$. Attention, la *longueur* d'un chemin fait référence à son nombre d'arêtes. Un chemin où chaque sommet n'apparaît qu'une seule fois dans la séquence (absence de boucle) est appelé *chemin simple*. Ceci implique que $u_i \neq u_j$, $1 \leq i < j \leq n$. Le sous-graphe du graphe du métro induit par les sommets $\{$`Ménilmontant, Couronnes, Belleville, Goncourt, République, Parmentier`$\}$ est un chemin simple à 6 sommets, de longueur 5.

Un *cycle* est un chemin où le premier et le dernier sommet de la séquence sont confondus, c'est-à-dire $u_n = u_1$. Le sous-graphe du graphe du métro induit par les sommets $\{$`Ménilmontant, Couronnes, Belleville, Goncourt, République, Parmentier, Rue St-Maur, Père Lachaise, Ménilmontant`$\}$ est un cycle.

Un graphe non-orienté est dit *connexe* s'il existe un chemin entre chaque paire de sommets du graphe. Par exemple, le graphe du métro Figure 1.1 n'est pas connexe car il existe au moins deux sommets (par exemple `Châtelet` et une des stations du funiculaire de Montmartre) sans chemin les reliant. En revanche, le graphe Figure 1.3 est

connexe. Une *composante connexe* d'un graphe G est un sous-graphe maximal connexe de G. Le graphe du métro Figure 1.1 contient deux composantes connexes, celle composée des deux stations du funiculaire de Montmartre et celle contenant toutes les autres stations.

Dans le cas d'un graphe orienté, un graphe est *faiblement connexe* si le remplacement des arcs par des arêtes produit un graphe connexe. Il est *fortement connexe* si pour chaque paire de sommets u et v, il existe un chemin orienté de u vers v et un chemin orienté de v vers u. Le graphe de la ligne 7bis Figure 1.2 est fortement connexe.

C'est certainement le mathématicien suisse Leonhard Euler (1707 – 1783) qui a introduit le premier la notion de graphe. En effet, en 1736, dans un article en latin ([Eul36], traduction en anglais dans [BLW86]) reprenant la démonstration faite l'année précédente devant les membres de l'Académie de Saint-Pétersbourg, il résout un problème populaire concernant les ponts de la ville prussienne de Königsberg (maintenant Kaliningrad en Russie). En effet, cette ville est construite autour de deux îles, et reliée à ces dernières *via* 7 ponts comme illustré ci-dessous. L'histoire raconte que les habitants ont longtemps essayé de trouver une route traversant tous les ponts exactement une seule fois. Euler a montré qu'une telle route n'existait pas. En effet, le (multi)graphe associé à cette configuration (illustré à droite du schéma ci-dessous), où chaque rive est un sommet et où chaque pont est une arête, contient plus de deux sommets de degré impair. Or, une telle route n'existe que si au plus deux sommets ont un degré impair. Si un graphe admet un chemin passant par chaque sommet du graphe et revenant au point de départ, il est dit *Eulérien*, en l'honneur d'Euler. Chaque sommet doit alors avoir un degré pair. Ceci se comprend intuitivement : pour chaque sommet, il faut pouvoir y arriver et en repartir.

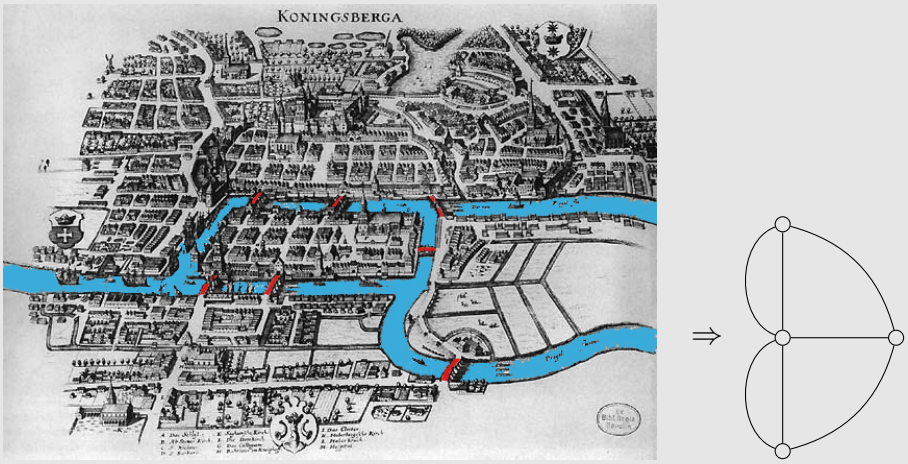


Illustration provenant de [BLW86].

Un *arbre* est un graphe non-orienté connexe, sans cycle. Le graphe correspondant

à la ligne 7 (Figure 1.3) est un arbre. Si le graphe non-orienté est sans cycle mais non connexe, c'est une *forêt*. C'est par exemple le cas du sous-graphe induit par les stations des lignes 7 et 13 du métro.

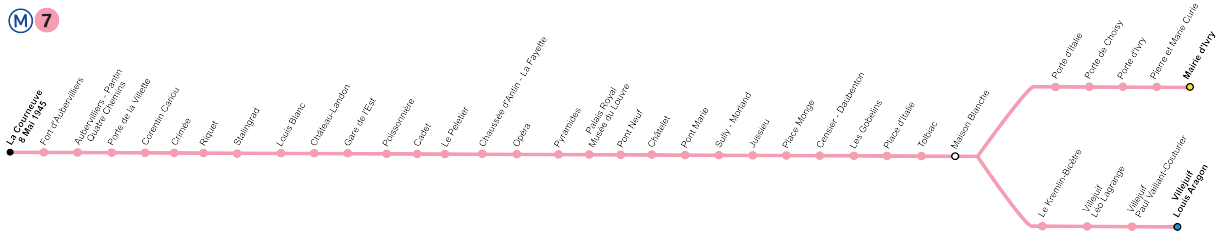


FIGURE 1.3 – Arbre représentant la ligne 7 du métro de Paris. Source Wikipedia.

Un *arbre enraciné* contient un sommet distinct des autres, appelé *racine*. Nous considérons dans la suite l'arbre de la Figure 1.3 enraciné à la station `La Courneuve 8 mai 1945`. Étant donné un nœud v d'un arbre enraciné, il existe un unique chemin allant de la racine à v. Chaque nœud u étant sur ce chemin est appelé *ancêtre* de v, alors que v est un *descendant* de u. Un *sous-arbre enraciné* en v est l'arbre composé de la racine v ainsi que de ses descendants. Par exemple, le sous-arbre enraciné au sommet `Maison Blanche` Figure 1.3 contient les nœuds `Porte d'Italie`, `Porte de Choisy`, `Porte d'Ivry`, `Pierre et Marie Curie`, `Mairie d'Ivry`, `Le Kremlin-Bicêtre`, `Villejuif Léo Lagrange`, `Villejuif Paul Vaillant-Couturier` et `Villejuif - Louis Aragon`. Si la dernière arête du chemin allant de la racine d'un arbre enraciné vers v est $\{u, v\}$, alors u est le *père* de v et v est le *fils* de u. Dans un arbre enraciné, la racine est le seul nœud sans père. Deux nœuds ayant le même père sont *frères* et les nœuds sans fils sont les *feuilles* de l'arbre. Les nœuds qui ne sont pas des feuilles sont des *nœuds internes*. Le sous-arbre enraciné à la station `Maison Blanche` de la ligne 7 contient deux feuilles, `Mairie d'Ivry` et `Villejuif - Louis Aragon`, tandis que `Porte d'Italie` et `Le Kremlin-Bicêtre` sont frères.

La longueur du chemin entre un nœud v d'un arbre enraciné et sa racine est la *profondeur* de v dans l'arbre enraciné. La longueur du plus long chemin entre v et une feuille est la *hauteur* de v dans l'arbre enraciné. La hauteur d'un arbre enraciné est la hauteur de sa racine. L'arbre enraciné à la station `La Courneuve 8 mai 1945` représentant la ligne 7 du métro est de hauteur 33, tandis que la station `Stalingrad` est de profondeur 7.

Un arbre enraciné est dit *binaire* si chaque nœud possède au plus deux fils. L'arbre enraciné à la station `La Courneuve 8 mai 1945` de la ligne 7 est binaire. Un arbre enraciné de hauteur h est dit *binaire complet* si tous les nœuds internes ont exactement deux fils et toutes les feuilles ont une profondeur de h ou $h - 1$.

Nous appelons *arbre orienté* un graphe orienté, faiblement connexe et sans cycle. Si le graphe orienté est sans cycle mais non connexe, c'est un Graphe Orienté Acyclique (ou *DAG* pour *Directed Acyclic Graph*).

Un *arbre orienté enraciné* est faiblement connexe et contient une unique *racine*. Ainsi, tous les nœuds de l'arbre enraciné exceptée la racine ont un unique père.

Les graphes ont d'autres applications, comme celui des collaborations scientifiques, où les sommets représentent des chercheurs et une arête existe si les deux chercheurs ont co-écrit un article. Remarquons qu'il existe dans ce graphe un chemin de longueur 8 entre Stéphane Vialette et l'actrice Natalie Portman (qui a publié un article), *via* Paul Erdös, mathématicien hongrois ayant écrit 1400 articles avec plus de 500 auteurs différents.

1.2 Problèmes et algorithmes

Un *algorithme* est un ensemble de règles, prenant des valeurs en entrée (une instance) et donnant des valeurs en sortie. Il est construit selon une suite finie d'étapes de calculs, transformant l'entrée en sortie. L'algorithme résout un *problème*. Le problème énonce une entrée, ainsi que sa relation souhaitée avec la sortie.

Donnons rapidement un exemple pour illustrer le lien entre problème et algorithme. Considérons le problème décrit dans la suite.

EXISTENCE D'UN ENTIER DANS UN TABLEAU

- **Entrée** : Un tableau de n entiers, un entier a.
- **Sortie** : Est-ce que a apparaît dans le tableau ?

Nous donnons l'Algorithme 1 pour le résoudre.

Algorithme 1 : ExistenceTableau

Entrées : Un tableau T de n entiers, un entier a

```
1 pour chaque i de 1 a n faire
2     si T[i] = a alors
3         retourner Oui
4 retourner Non
```

Il s'agit bien d'une suite finie d'opérations, effectuant une comparaison entre chaque élément du tableau donné en entrée et l'entier a, et fournissant en sortie une donnée répondant au problème, ici Oui si a est contenu dans le tableau, Non sinon (l'ensemble du tableau a été parcouru sans trouver l'élément recherché). Remarquons qu'un algorithme n'est pas un *programme informatique*. L'implémentation d'un algorithme selon

un langage de programmation donne un programme. Le choix d'un langage est sans importance dans l'élaboration et l'énonciation d'un algorithme.

Dans la suite de cette section, nous décrivons les méthodes de classification d'algorithmes et de problèmes.

1.2.1 Classification et analyse d'algorithmes

Le terme algorithme ne vient *pas* du grec ancien *algos* signifiant douleur, mais d'une latinisation du nom du mathématicien perse Al-Khawarizmi (783 – 850).

Un algorithme défini pour un problème donné est *correct* s'il résout ce dernier. Nous voyons assez simplement que l'Algorithme 1 résout le problème EXISTENCE D'UN ENTIER DANS UN TABLEAU. Il est cependant parfois plus délicat de prouver qu'un algorithme est effectivement correct. Contrairement à l'intuition première, la conception d'un algorithme incorrect peut parfois être utile, comme nous le verrons Section 1.3.4.

L'efficacité d'un algorithme se juge en général selon deux critères :

1. Son *temps d'exécution*. Plus l'algorithme met de temps à se terminer, moins il est considéré comme efficace.
2. L'*espace mémoire qu'il utilise*. Plus l'algorithme utilise de mémoire, moins il est considéré comme efficace.

Notons que l'optimisation de ces deux critères est généralement antagoniste. En effet, il est courant d'augmenter l'espace mémoire utilisé afin de diminuer la durée d'exécution (en stockant un ensemble de résultats précédemment calculés par exemple).

Considérons maintenant comme exemple l'algorithme pour chanter la célèbre comptine « Un kilomètre à pied » (voir Algorithme 2).

Algorithme 2 : KilometreAPied

Entrées : Un nombre n de kilomètres

```
1 Chanter (1 kilomètre à pied, ça use, ça use,);
2 Chanter (1 kilomètre à pied, ça use les souliers.);
3 pour chaque i de 2 a n faire
4     Chanter (i kilomètres à pied, ça use, ça use,);
5     Chanter (i kilomètres à pied, ça use les souliers.);
```

Le temps de chanter cette chanson pour un nombre n de kilomètres dépend bien sûr de la vitesse de chant. Mais il dépend essentiellement du nombre de kilomètres.

Ainsi, chanter KilometreAPied($2n$) prendra deux fois plus de temps que de chanter KilometreAPied(n) quelle que soit la vitesse de chant.

De manière similaire, nous voulons pouvoir mesurer l'efficacité d'un algorithme (en terme de durée d'exécution) de manière indépendante de la vitesse du processeur et du langage de programmation utilisé pour implémenter l'algorithme. Nous utilisons pour cela la notion de *complexité*. La complexité est le nombre d'opérations élémentaires (indivisibles) nécessaires à l'accomplissement de l'algorithme. Nous exprimons généralement ce nombre en fonction de la taille de l'entrée. Dans l'exemple de l'Algorithme 2, quel que soit la vitesse de chant, il faudra chanter n fois les deux phrases.

Prenons maintenant pour exemple l'algorithme pour chanter cette autre comptine enfantine, « Alouette » (voir Algorithme 3).

Algorithme 3 : Alouette

Entrées : Tableau $PartiesDuCorps$ de n chaînes de caractères

```
1 Chanter (Alouette, gentille alouette, alouette, je te plumerai.);
2 pour chaque i de 1 a n faire
3     Chanter (Je te plumerai PartiesDuCorps[i]. Je te plumerai
      PartiesDuCorps[i]);
4     pour chaque j de i a 1 faire
5         Chanter (Et PartiesDuCorps[j] ! Et PartiesDuCorps[j] !);
6     Chanter (Alouette ! Alouette ! Oooo !);
7     Chanter (Alouette, gentille alouette, alouette, je te plumerai.);
```

Le nombre de fois où le chanteur chantera la ligne 5 est égal à $\sum_{i=1}^{n} i$ (il la chante 1 fois lorsqu'une seule partie du corps a été évoquée, 2 fois lorsque deux parties ont été évoquées, 3 fois lorsque trois parties ont été évoquées, et ainsi de suite). Ceci est égal à $\frac{n(n+1)}{2} \leq n^2$. Cela veut dire que pour un n donné, chanter Alouette(n) prendra environ n fois plus de temps que de chanter KilometreAPied(n).

L'analyse de complexité la plus populaire est celle de l'analyse de la *complexité dans le pire des cas*. Autrement dit, déterminer la complexité pire cas consiste à calculer le nombre d'opérations nécessaires pour résoudre un problème dans le pire des cas. Pour résoudre le problème EXISTENCE D'UN ENTIER DANS UN TABLEAU, dans le pire des cas, l'entier recherché se trouve dans la dernière case du tableau de taille n ou est absent du tableau : il faut alors n opérations de comparaisons pour répondre au problème. C'est cependant les seuls cas où il faut effectivement n opérations. Par exemple, si l'entier recherché se trouve en première case, une seule opération est nécessaire.

Dans le cadre de cette thèse, nous nous intéressons exclusivement à l'analyse dans le pire cas. Cependant, il existe d'autres analyses parmi lesquelles nous citons :

- *l'analyse en moyenne*, estimant le nombre d'opérations nécessaires en moyenne pour une instance, selon une distribution de probabilité donnée,
- *l'analyse amortie* (*amortized*), estimant la somme des opérations nécessaires pour

une séquence d'instances,
- *l'analyse lissée* (*smoothed*), estimant le nombre d'opérations nécessaires pour une instance pire cas perturbée aléatoirement.

L'analyse du comportement moyen d'un algorithme semble mieux refléter la véritable complexité du problème. Cependant, son étude requiert une analyse complexe ainsi qu'une distribution de probabilité.

Donner le nombre exact d'opérations élémentaires nécessaires pour effectuer l'algorithme peut-être sans intérêt et prêter à confusion. Ce nombre d'opérations peut varier selon certaines valeurs spécifiques de l'entrée. Il peut aussi dépendre des détails d'implémentations, créant un nombre constant d'opérations en plus à effectuer. Enfin, indiquer que dans le pire des cas la complexité de l'algorithme est de $f(n) = 45n^2+7514n+45\lg(n)$ présente peu d'intérêt (en plus d'être difficile à calculer). Il faut juste remarquer ici que le temps de calcul augmente quadratiquement avec n, car $45n^2$ est le terme de plus haut degré, et 45 est un facteur constant peu intéressant. Pour ces raisons, il est admis l'utilisation des *notations asymptotiques* (*Big Oh* en anglais), définies ci-après, nous permettant de dire que $f(n) = \mathcal{O}(n^2)$ (voir aussi Figure 1.4).

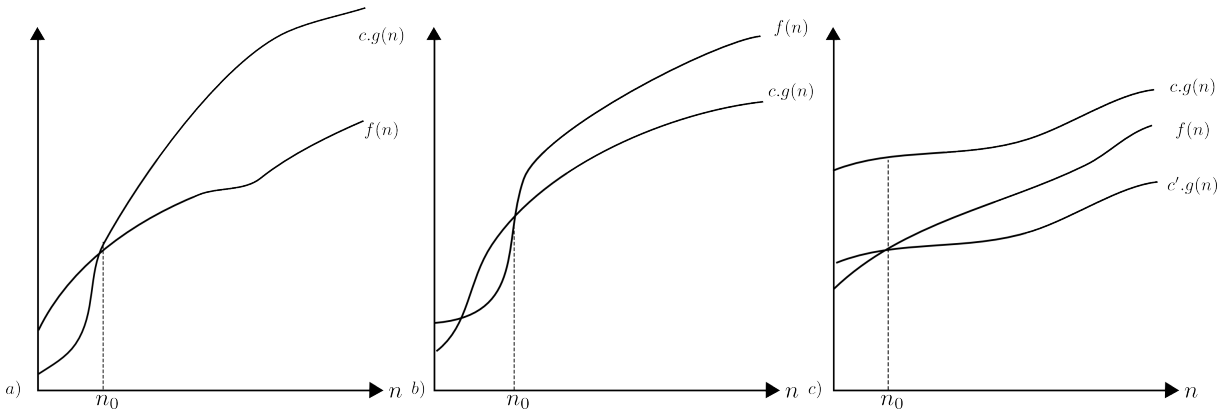


FIGURE 1.4 – Illustration des notations asymptotiques. À partir de n_0, il existe des constantes c et c' telles que la fonction f en a) est dominée par $c.g(n)$, donc $f = \mathcal{O}(g)$, en b) domine $c.g(n)$, donc $f = \Omega(g)$, en c) est dominée par $c.g(n)$ et domine $c'.g(n)$, donc $f = \Theta(g)$.

Considérant deux fonctions f et g, alors :
- $f = \mathcal{O}(g)$ s'il existe une constante c telle que $f(n) \leq c.g(n)$, pour un n suffisamment grand (c'est-à-dire $n \geq n_0$ avec n_0 une constante),
- $f = \Omega(g)$ si $g = \mathcal{O}(f)$,
- $f = \Theta(g)$ si $f = \mathcal{O}(g)$ et $g = \mathcal{O}(f)$,
- $f = o(g)$ si pour chaque $\epsilon > 0, f(n) \leq \epsilon.g(n)$, pour un n suffisamment grand,
- $f = \omega(g)$ si $g = o(f)$.

Intuitivement, la notation $f = \mathcal{O}(g)$ exprime que f est plus petite ou égale que g si les facteurs constants sont ignorés. Par exemple, si $f(n) = 100n^2+10n$ et $g(n) = n^2$, alors $f = \mathcal{O}(g)$, que nous écrivons $f(n) = \mathcal{O}(n^2)$. Remarquons qu'il y a également $g = \mathcal{O}(f)$, et donc $f = \Theta(g)$ (et $g = \Theta(f)$). Vérifions qu'il est possible de trouver une constante

c telle que $f(n) \leq c.g(n^2)$ pour tous les n suffisamment grands. En effet, en prenant $c = 102$ et $n_0 = 5$, pour tout $n > 5$, nous avons bien $100n^2 + 10n \leq 102n^2$. Attention, $f = \mathcal{O}(n^{10})$ est aussi vrai. Par contre, $f = \mathcal{O}(n)$ n'est pas vrai. En effet, il est impossible de trouver c et n_0 tels que $f(n) \leq c.n, \forall n \geq n_0$. Lorsqu'on emploie cette notation, nous ne donnons pas les valeurs de c et n_0 pour lesquelles la borne est vraie. Ces valeurs sont d'ailleurs différentes à chaque utilisation de cette notation.

Dans cette thèse, nous utiliserons presque exclusivement la notation $\mathcal{O}$. Pour plus de détails et d'exemples, le lecteur pourra se référer par exemple au Chapitre 3 de [CLRS02] ou au Chapitre 2 de [Ski08].

Les notations asymptotiques sont également appelées notations de Landau, du nom du mathématicien Allemand Edmund Landau (1877 – 1938). Knuth pointe les origines de ces notations dans une lettre aux éditeurs de SIGACT [Knu76]. Il en profite pour recommander leurs usages et regretter l'usage abusif de $\mathcal{O}$ quand Θ ou Ω sont plus adaptés.

La notion de complexité en espace existe également. Alors, la taille nécessaire à l'algorithme pour s'exécuter est bornée asymptotiquement. Par la suite, lorsque le terme de complexité sans précision supplémentaire sera employé, il s'agira de la complexité en temps.

Certains types de complexités sont nommés de la manière suivante, où n est la taille de l'entrée :

- Les algorithmes en temps *constant* ont une complexité de type $\mathcal{O}(1)$. Le nombre d'opérations reste constant, indépendamment de la taille de l'instance.
- Les algorithmes *polynomiaux* ont une complexité de type $\mathcal{O}(n^c)$, où c est une constante. Certains algorithmes polynomiaux sont nommés spécifiquement :
  - Les algorithmes *logarithmiques* ont une complexité de type $\mathcal{O}(\log(n))$. La complexité croît légèrement avec n. Ce type de complexité se rencontre dans une boucle où la taille de l'instance est divisée à chaque itération.
  - Les algorithmes *linéaires* ont une complexité de type $\mathcal{O}(n)$. C'est typiquement la complexité d'une boucle effectuant n itérations, avec un nombre constant d'opérations à chaque itération, comme pour l'Algorithme 2.
  - Les algorithmes *quadratiques* ont une complexité de type $\mathcal{O}(n^2)$. C'est la complexité de deux boucles imbriquées : un boucle effectue n fois une boucle effectuant au plus n itérations. C'est le cas de l'Algorithme 3.
- Les algorithmes *exponentiels* ont une complexité de type $\mathcal{O}(c^n)$, où c est une constante. Par exemple, énumérer tous les sous ensembles de l'ensemble $\{1, 2, \ldots, n\}$ s'effectue en $\mathcal{O}(2^n)$.

n \ $f(n)$	n	n^2	2^n	$n!$
10	< 1 sec	< 1 sec	< 1 sec	< 1 sec
20	< 1 sec	< 1 sec	< 1 sec	77.1 ans
30	< 1 sec	< 1 sec	1 sec	8.4×10^{15} ans
40	< 1 sec	< 1 sec	18.3 min	
50	< 1 sec	< 1 sec	13 jours	
100	< 1 sec	< 1 sec	4×10^{13} ans	
1,000	< 1 sec	< 1 sec		
10,000	< 1 sec	< 1 sec		
100,000	< 1 sec	10 sec		
1,000,000	< 1 sec	16.7 min		
10,000,000	< 1 sec	1.16 jours		
100,000,000	< 1 sec	115.7 jours		
1,000,000,000	1 sec	31.7 ans		

TABLE 1.1 – Temps estimé pour effectuer n opérations pour un algorithme de complexité $f(n)$, en considérant un processeur (assez ancien) prenant 10^{-9} secondes pour effectuer une opération. Les valeurs de ce tableau proviennent de [Ski08].

Comme montré sur la Table 1.1, il est important de trouver un algorithme avec la plus faible complexité possible. En effet, si tous les algorithmes sont envisageables en pratique pour des entrées de taille inférieure à 10, les algorithmes exponentiels ne sont pas envisageables pour une implémentation avec des entrées de taille supérieure à 40. De plus, dire qu'un algorithme, actuellement non praticable, le sera quelques années plus tard grâce aux améliorations technologiques reste un mauvais pari. En effet, si, augmenter la vitesse de calcul a un effet multiplicatif sur la taille des instances gérées par un algorithme polynomial, c'est un effet additif sur un algorithme de complexité exponentielle (voir Table 1.2).

Dans la sous-section suivante, nous montrons qu'il est parfois possible de prouver que trouver un algorithme exact avec une complexité polynomiale pour un problème donné est impossible, tant qu'une conjecture (très) probable n'est pas montrée fausse.

1.2.2 Classification de problèmes

Nous voyons dans cette sous-section ce que sont les classes P et NP et comment montrer qu'un problème appartient à l'une ou l'autre de ces classes.

Un problème peut être soit de *décision*, soit d'*optimisation*. Dans le premier cas, la

$f(n)$	Avec un ordinateur actuel	Avec un ordinateur 100 fois plus rapide	Avec un ordinateur 1,000 fois plus rapide
n	N_1	$100N_1$	$1000N_1$
n^2	N_2	$10N_2$	$31.6N_2$
n^3	N_3	$4.64N_3$	$10N_3$
2^n	N_4	$N_4 + 6.64$	$N_4 + 9.97$
3^n	N_5	$N_5 + 4,19$	$N_5 + 6.29$

TABLE 1.2 – Tailles des instances maximales pouvant être gérées par un ordinateur pour une durée donnée, en fonction de la complexité ($f(n)$) de son algorithme. Les données de ce tableau proviennent de [GJ79].

réponse sera *Oui* ou *Non*. Dans le second cas, la réponse attendue doit optimiser, parmi l'ensemble des solutions possibles, une certaine propriété donnée par le problème. Le type d'optimisation peut être précisé. Il peut s'agir d'un problème de *minimisation*, ou de *maximisation*. Illustrons ces deux types de problèmes avec respectivement la version d'optimisation et de décision du problème de voyageur de commerce.

MINIMUM TRAVELLING SALESMAN PROBLEM

• **Entrée** : Un ensemble de n villes $\{v_1, v_2, \ldots, v_n\}$, la distance $d(v_i, v_j)$ entre deux villes v_i et v_j pour chaque paire $v_i \neq v_j$.

• **Sortie** : Un chemin passant par toutes les villes une et une seule fois.

• **Mesure** : La taille du chemin.

TRAVELLING SALESMAN PROBLEM

• **Entrée** : Un ensemble de n villes $\{v_1, v_2, \ldots, v_n\}$, la distance $d(v_i, v_j)$ entre deux villes v_i et v_j pour chaque paire $v_i \neq v_j$ et un entier k.

• **Sortie** : Existe t-il un chemin de longueur inférieure à k passant par toutes les villes une et une seule fois ?

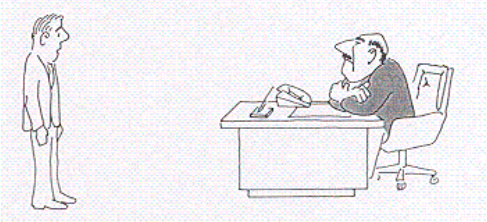
Beaucoup de problèmes naturels sont des problèmes d'optimisation. Il est cependant assez facile de leur donner une version de décision en ajoutant un paramètre dans l'entrée (ici, la longueur finale du chemin).

Si nous pouvons résoudre le problème d'optimisation, nous pouvons résoudre le problème de décision associé : il suffit de comparer la valeur de retour du problème d'optimisation avec le paramètre du problème de décision (ici, si le problème d'optimisation donne une solution de 10 kilomètres, on peut répondre Oui au problème de

décision demandant un chemin de longueur inférieure à k si $k \geq 10$, et Non si $k < 10$). Inversement, si nous pouvons résoudre le problème de décision, nous pouvons trouver le paramètre donnant la solution optimale en testant pas à pas chaque paramètre.

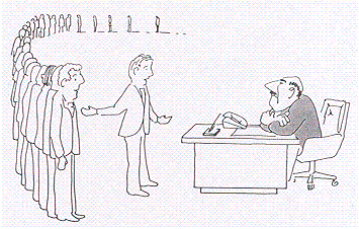
Un problème se trouve dans la classe P s'il existe un algorithme qui le résout avec une complexité polynomiale, c'est-à-dire en temps $\mathcal{O}(n^c)$, où c est une constante et n la taille de l'entrée, comme vu précédemment. Pour prouver l'appartenance d'un problème à cette classe, il suffit donc d'exhiber un algorithme polynomial donnant une solution exacte à ce problème.

Toutefois, il est souvent impossible de trouver un algorithme polynomial de ce type, même pour un brillant chercheur (voir Figure 1.5). Il est alors possible de montrer que trouver un tel algorithme est une tâche *a priori* impossible, en montrant que le problème est NP-Complet.



a) « Je n'arrive pas à trouver d'algorithme efficace pour ce problème, je suppose que je suis trop bête... »

b) « Je n'arrive pas à trouver d'algorithme efficace pour ce problème car ce n'est pas possible d'en trouver un ! »



c) « Je n'arrive pas à trouver d'algorithme efficace pour ce problème, comme aucune de ces personnes reconnues ! »

FIGURE 1.5 – Dessin très connu, publié dans [GJ79]. L'histoire associée est la suivante : votre patron vous demande un algorithme efficace pour un problème NP-Complet. Après avoir tenté pendant des semaines de trouver un tel algorithme, vous ne voulez pas avoir à revenir voir votre patron et devoir dire que vous avez échoué (a). Pour garder votre emploi, il faudrait mieux pouvoir lui dire qu'un tel algorithme est introuvable (b). Il serait encore mieux de pouvoir lui annoncer qu'embaucher quelqu'un d'autre n'y changerait probablement rien (c).

La classe NP contient l'ensemble des problèmes où, étant donnée une instance de ce problème, il est possible de déterminer en temps polynomial si celle-ci est une solution pour le problème – nous sommes capables de donner un certificat. Il est alors évident que $\mathsf{P} \subseteq \mathsf{NP}$ car un algorithme polynomial est également un certificat. Déterminer si $\mathsf{NP} \subseteq \mathsf{P}$ (et par conséquent si $\mathsf{P} = \mathsf{NP}$ ou $\mathsf{P} \neq \mathsf{NP}$) est par contre une célèbre question ouverte depuis les années 1970 – une récompense d'1 million de dollars est promise par

le Clay Mathematical Institute à celui qui donnera une réponse à ce problème [1]. Cette récompense motive des recherches plus ou moins sérieuses, et au moins 70 articles (non évalués par des pairs) prétendent y avoir répondu [2]. La plupart des théoriciens pensent néanmoins que $\mathsf{P} \neq \mathsf{NP}$.

Une définition équivalente de la classe NP indique que les problèmes de la classe NP peuvent être résolus en temps polynomial par un algorithme non déterministe (donc qui n'exécute pas toujours la même suite d'opérations pour une même entrée). Le nom NP vient de cette définition (Nondeterministic Polynomial time).

La Section 5.2 de [GJ79] donne l'histoire des choix des terminologies autour de NP. Ceux-ci ont en fait été choisis par la communauté scientifique de l'époque, suite à un vote privé organisé par Knuth. L'acronyme PET pour *Probably Exponential Time* (Probablement en Temps Exponentiel) avait été proposé de manière humoristique. Cet acronyme avait l'avantage de pouvoir signifier *Provably Exponential Time* (Temps Exponentiel Prouvé) si $\mathsf{P} \neq \mathsf{NP}$ et *Previously Exponential Time* (Auparavant Temps Exponentiel) si $\mathsf{P} = \mathsf{NP}$.

Un problème P_1 se réduit polynomialement en un problème P_2 si pour *toute* instance I_1 de P_1, nous pouvons construire *une* instance I_2 de P_2 en temps polynomial à l'aide d'une fonction f, tel que I_1 est vraie si et seulement si $I_2 = f(I_1)$ est également vraie.

Ainsi, si nous pouvons résoudre P_2 en temps polynomial, alors nous pouvons résoudre P_1 en temps polynomial. En effet, depuis l'instance de P_1, nous transformons polynomialement celle-ci en une instance de P_2, que l'on résout en temps polynomial. Comme cette instance est vraie si et seulement si l'instance de P_1 est vraie, nous avons un algorithme polynomial (en deux parties) qui résout P_1 (voir Figure 1.6).

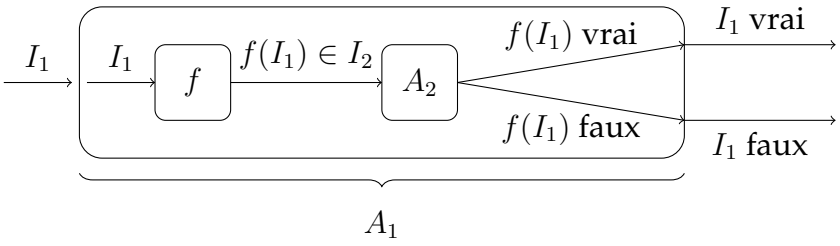


FIGURE 1.6 – Illustration d'un problème P_1 se réduisant à P_2. L'algorithme A_1 résout P_1 de la manière suivante. Une instance I_1 de P_1 est en entrée de l'algorithme A_1. Nous réduisons cette instance à une instance de P_2 avec la fonction f. L'algorithme A_2 résout les instances I_2 de P_2, donc l'instance transformée $f(I_1)$. Ensuite, $f(I_1)$ est vrai si et seulement si I_1 est vrai.

Un problème est NP-Difficile s'il se réduit polynomialement à n'importe quel autre

1. http://www.claymath.org/millennium/
2. http://www.win.tue.nl/~gwoegi/P-versus-NP.htm

problème de la classe NP. Autrement dit, il est au moins aussi difficile que n'importe quel autre problème de la classe NP.

Les problèmes dits NP-Complets sont considérés comme les plus difficiles de la classe NP. Un problème est NP-Complet s'il réunit les deux conditions suivantes (voir aussi Figure 1.7) :

– il appartient à la classe NP,
– il est NP-Difficile.

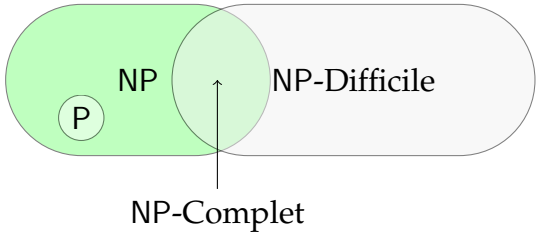


FIGURE 1.7 – La relation entre les ensembles P, NP, NP-Complet et NP-Difficile, en supposant vraie la conjecture P $\neq$ NP.

Selon la stricte définition, un problème d'*optimisation* NP-Difficile ne peut pas être considéré NP-Complet, car nous ne pouvons pas dire qu'il appartient à la classe NP. En effet, il est impossible de décider en temps polynomial si une instance est une solution, car il faudrait pouvoir déterminer en temps polynomial si cette instance est la meilleure. Ceci est impossible si le problème est NP-Difficile. Cette confusion se trouve parfois dans la littérature. Cependant, cette erreur n'est pas importante car nous avons vu qu'il était possible de transformer un problème d'optimisation en problème de décision.

Comme dit précédemment, il est largement admis que P $\neq$ NP. Ainsi, lorsqu'un problème est prouvé NP-Complet (après avoir prouvé son appartenance à NP et montré qu'il se réduisait polynomialement à n'importe quel problème de NP), nous pouvons supposer qu'il n'existe pas d'algorithme polynomial pour le résoudre. En pratique, plutôt que de montrer qu'un problème se réduit polynomialement à tous les problèmes de NP, nous pouvons de manière équivalente prouver qu'il se réduit à un problème NP-Complet, car la définition de réduction polynomiale est transitive (si P_1 se réduit à P_2 et que P_2 se réduit à P_3, alors P_1 se réduit à P_3).

Ainsi, pour prouver qu'un problème est NP-Difficile, il faut le réduire à un problème connu NP-Difficile. Ceci est possible car il existe au moins un problème NP-Complet : il s'agit du problème SAT.

BOOLEAN SATISFIABILITY (SAT)

• **Entrée** : Un ensemble X de variables booléennes (c'est-à-dire $\in \{0, 1\}$), un ensemble C de clauses, où les clauses sont des disjonctions (OU logique) de littéraux (un littéral est une variable de X ou sa négation).

• **Sortie** : Une affectation des variables de X telle que toutes les clauses de C sont rendues vraies par cette affectation.

C'est le premier problème prouvé NP-Complet par Cook [Coo71] en 1971 (un résultat similaire a été montré indépendamment par Levin [Lev73]). La preuve de Cook est présentée de manière simplifiée dans [GJ79]. L'année suivante, Karp [Kar72] prouve que plusieurs problèmes (dont le problème TRAVELLING SALESMAN PROBLEM) sont aussi difficiles que le problème SAT. Depuis, un grand nombre (des milliers !) de problèmes ont été prouvés NP-Complets. Si une personne trouvait un algorithme polynomial pour l'un d'entre eux, cela impliquerait un algorithme polynomial pour tous les problèmes NP-Complets. Des techniques pour effectuer des réductions polynomiales ainsi que des exemples de problèmes NP-Complets se trouvent dans le livre de référence de Garey et Johnson [GJ79].

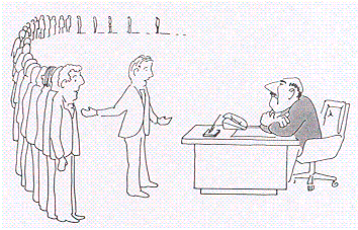
Beaucoup de puzzles, jeux de solitaire et jeux à deux joueurs intéressants sont NP-Difficiles (peut-être la raison de leur intérêt !). Citons par exemple le Démineur, Tétris, Sudoku, Othello, Go, ou les échecs [DH09].

1.3 Contourner la NP-Complétude

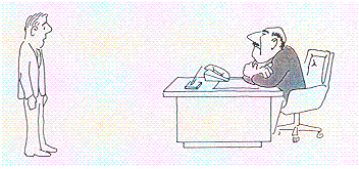
S'il est bien entendu utile de prouver qu'un problème est NP-Complet, il est souvent peu satisfaisant de rester sur ce fait (Figure 1.8). En effet, beaucoup de problèmes ayant une application pratique s'avèrent être NP-Complets. Nous montrons dans cette section que tout n'est pas perdu – beaucoup de possibilités s'ouvrent alors pour, tout de même, résoudre le problème. Nous pouvons globalement diviser ces possibilités en deux catégories : certaines techniques donneront une réponse exacte au problème dans un temps dit acceptable, d'autres donneront rapidement une réponse au problème, sans que celle-ci soit toutefois exacte. Avant de montrer ces techniques, nous nous intéressons à observer des sous-problèmes, acceptant possiblement un algorithme polynomial.

Toutes les instances d'un problème NP-Complet ne sont pas forcément difficiles. Au contraire, il peut exister des problèmes où la majorité des instances peuvent être résolues en temps polynomial – la difficulté du problème restant confinée à des cas pathologiques. Par exemple, la plupart des jeux d'essais disponibles pour le problème SAT sont résolus en quelques minutes par les solveurs actuels. Il existe une compétition internationale visant à trouver des jeux d'essais pertinents pour pouvoir comparer ces solveurs[a].
Dans le cas de problèmes modélisés depuis la biologie, il est possible que les instances pertinentes biologiquement d'un problème prouvé NP-Complet puissent être résolues en temps polynomial – les instances difficiles concernant des cas n'apparaissant pas ou peu dans des données réelles. Il reste donc indispensable d'être capable de déterminer les instances d'un problème pouvant être résolues en temps polynomial.

a. `http://www.satcompetition.org/`



a) « Je n'arrive pas à trouver d'algorithme efficace pour ce problème, comme aucune de ces personnes reconnues ! »



b) « Et alors ?! J'ai besoin d'une solution maintenant ! »

FIGURE 1.8 – Adaptation du dessin Figure 1.5 de Garey et Johnson par Danny Hermelin [Her09].

1.3.1 Analyser des sous-problèmes

Si un problème est NP-Complet, rien n'empêche qu'une instance moins difficile puisse être dans la classe P. Nous pouvons alors essayer de déterminer quelles sont les contraintes additionnelles nécessaires sur l'entrée d'un problème NP-Complet pour que celui-ci soit dans P.

Par exemple, pour un problème NP-Complet prenant en entrée un graphe, limiter le degré de ce dernier peut être une condition suffisante pour que le problème soit dans P. Ainsi, beaucoup de problèmes sur les graphes sont polynomiaux si le degré du graphe est restreint à 2. Prenons pour exemple, le problème suivant :

VERTEX COVER

• **Entrée** : Un graphe $G = (V, E)$, un entier k.

• **Sortie** : Existe-il un sous-ensemble $V' \subseteq V$, avec $|V'| \leq k$, tel que pour chaque arête $\{u, v\} \in E, \{u, v\} \cap V' \neq \emptyset$?

Le problème VERTEX COVER est dans P si le graphe d'entrée est de degré 2, mais NP-Complet si le degré de G est supérieur ou égal à 3. Dans ce cas, la borne est la plus fine possible puisque nous connaissons exactement le moment où le problème bascule dans la difficulté en raison du degré du graphe. Nous verrons aussi, dans la suite de cette thèse, que certains problèmes NP-Complets sur les graphes, sont dans P lorsqu'on considère plutôt un arbre ou un chemin. La Section 4.1 de [GJ79] donne plus de détails en ce qui concerne l'analyse de sous-problèmes.

Nous avons vu dans cette section que nous pouvons ajouter des contraintes à un problème afin de pouvoir y répondre en temps polynomial. Cependant, nous pouvons aussi attendre une réponse à un problème selon sa définition originale dans un temps acceptable, ce que nous montrons par la suite.

1.3.2 La complexité paramétrée

Un algorithme de *complexité paramétrée* permet d'obtenir une réponse exacte, avec un temps de calcul dit acceptable. Elle fut introduite durant les années 1990, et les ouvrages de référence sont [DF99, Nie06, FG06] (le livre de Niedermeier est plus orienté sur la création d'algorithmes alors que celui de Flum et Grohe est plus orienté sur les aspects théoriques).

Plus précisément, l'entrée d'un *problème paramétré* est composée d'un objet X et d'un paramètre k. Le problème VERTEX COVER est un exemple de problème paramétré, où X est le graphe d'entrée et k la taille du sous-ensemble recherché.

Un problème, prenant un objet X et un entier k en entrée, appartient à la classe de complexité FPT (Fixed-Parameter Tractable) s'il existe un algorithme répondant à ce problème avec une complexité de la forme $f(k).|X|^c$, où c est une constante et f est une fonction quelconque ne dépendant que de k. En d'autre termes, l'algorithme est exponentiel (nous cherchons des algorithmes de complexité paramétrée pour des problèmes NP-Complets), mais la partie exponentielle dépend uniquement du paramètre k, et non de la taille de l'entrée ($|X|$). L'explosion combinatoire est restreinte à un paramètre spécifique, que nous savons (ou espérons) petit devant $|X|$ en pratique (voir Figure 1.9). Par exemple, un problème ayant un algorithme de complexité $\mathcal{O}(2^k.n^2)$ est dans FPT, n représentant la taille de l'entrée. Attention cependant, un algorithme ayant pour complexité $\mathcal{O}(2^{2^{2^{2^{2^k}}}} n)$ est également dans FPT, mais semble peu intéressant, même avec $k = 1$.

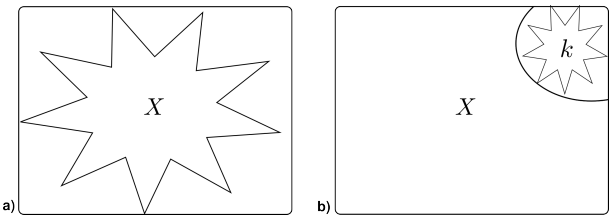


FIGURE 1.9 – Schéma proposé par Downey et Fellows [DF99]. En a), l'explosion combinatoire selon l'entrée X. En b), l'explosion combinatoire est confinée à un paramètre k, jugé petit par rapport à la taille de X.

Nous en profitons pour introduire une notation étendant les notations asymptotiques vues à la Section 1.2.1. En effet, dans le cadre de la complexité paramétrée, il est souvent d'usage d'utiliser la notation $\mathcal{O}^*(f(k))$, qui supprime les facteurs polynomiaux. Ainsi, si la complexité d'un algorithme est $\mathcal{O}(2^k n^2)$, nous pouvons aussi l'écrire $\mathcal{O}^*(2^k)$.

De manière similaire à la complexité classique, nous pouvons prouver qu'un problème n'appartient pas *a priori* à la classe FPT ; à savoir qu'il est *parameterized intractable*. Cet aspect sera développé dans la Section 1.3.2.4.

La complexité paramétrée offre une alternative à la difficulté d'un problème car nous supposons le paramètre k (beaucoup) plus petit que la taille de l'entrée. Ainsi, il est plus probable d'obtenir un temps d'exécution convenable pour des instances typiques. Il existe de nombreux domaines d'applications pour la complexité paramétrée. Par exemple, il est probable que la taille d'une requête k dans une base de données sera beaucoup plus petite que la taille n de cette dernière. En bio-informatique, nous recherchons des séquences ou structures d'intérêt de taille k inférieure à 20 dans des séquences ou réseaux de taille n beaucoup plus importante (plusieurs milliers). Le nombre de mouvements k dans un jeu est beaucoup plus petit que l'ensemble des mouvements possibles n. Si des algorithmes de complexité $\mathcal{O}(2^k n)$ existent pour ces problèmes, ils seront certainement utiles.

La *paramétrisation standard* d'un problème prend comme paramètre k, la taille de la solution (ce qui est le cas avec le k du problème VERTEX COVER). D'autres paramètres sont possibles – il est cependant parfois difficile de trouver un paramètre pertinent. En effet, le choix d'un paramètre trop grand, proche de la taille de l'entrée offre peu d'intérêt. En revanche, la recherche d'un algorithme de complexité paramétrée avec un paramètre trop petit peut être vouée à l'échec si le problème n'est pas dans P. Il est également courant de chercher une paramétrisation duale, c'est-à-dire un paramètre $k' = n - k$. Ce paramètre dual est alors pertinent si k est grand. Il est aussi possible de chercher selon un paramètre au-dessus d'une valeur garantie. Prenons pour exemple le problème suivant :

INDEPENDENT SET

• **Entrée** : Un graphe $G = (V, E)$, un entier k.

• **Sortie** : Un sous-ensemble des sommets $V' \subseteq V$ tel que $|V'| = k$ et V' est un ensemble indépendant, c'est-à-dire qu'il n'existe pas deux sommets $u, v \in V'$ tels que $\{u, v\} \in E$.

Nous savons qu'une solution au problème INDEPENDENT SET dans un graphe planaire est de taille supérieure à $\lceil \frac{n}{4} \rceil$, où n est le nombre de sommets dans le graphe. Par conséquent, le paramètre taille de la solution sera assez grand. Cependant, nous pouvons tenter de trouver un algorithme de complexité paramétrée pour répondre au problème (ouvert) « Existe t-il une solution au problème INDEPENDENT SET de taille $\lceil \frac{n}{4} \rceil + k$ lorsque G est un graphe planaire ? » [Fel01].

Il n'existe pas de méthode générale pour concevoir un algorithme de complexité paramétré. Il faut une étude précise du problème pour démontrer certaines propriétés de celui-ci. Cependant, quelques techniques génériques ont été développées ces dernières années. La suite de cette section en présente certaines, avant de montrer comment prouver qu'un problème n'appartient pas *a priori* à la classe FPT pour un paramètre donné.

1.3.2.1 Programmation dynamique et Color-coding

La technique de programmation dynamique n'est pas spécifique à la complexité paramétrée. Elle est utilisée de manière plus large, pour améliorer la complexité en temps de nombreux algorithmes, depuis des années. Le principe de la programmation dynamique est assez simple. Intuitivement, nous stockons le résultat de calculs dans une table afin de pouvoir les utiliser à nouveau si nécessaire et sans les recalculer. Ceci s'utilise souvent car de nombreux problèmes basés sur la récursivité nécessitent le calcul redondant d'une fonction avec les mêmes paramètres.

Si le terme *programmation dynamique* ne reflète pas vraiment la technique (il n'y pas de programmation, ni de dynamisme), c'est parce que Richard Bellman, son inventeur, voulait rendre séduisant son travail auprès de son responsable, connu pour son hostilité à la recherche mathématique. Il pensait en effet qu'il était impossible d'utiliser le mot dynamique dans un sens péjoratif [Edd04].

Considérons par exemple la suite de Fibonacci (notée par la suite $Fibo$), introduite par Fibonacci pour décrire la croissance d'une population idéale de lapins. Le calcul de $Fibo(n)$, $n > 2$ est égal à la somme de $Fibo(n-1)$ et $Fibo(n-2)$ (avec $Fibo(1) =$

$Fibo(2) = 1$). Constatons que le calcul de $Fibo(n-1)$ entraîne le calcul de $Fibo(n-2)$. Nous imaginons aisément que le stockage des calculs évite la redondance et accélère l'algorithme correspondant.

Plus généralement, la programmation dynamique est souvent utile quand une récurrence est nécessaire pour répondre à un problème, et est souvent utilisée en bioinformatique [Gus97]. Pour illustrer le principe du color-coding lié à la programmation dynamique, introduisons le problème suivant :

$k-$PATH

• **Entrée** : Un graphe $G = (V, E)$, un entier k.

• **Sortie** : Existe-il un chemin simple (c'est-à-dire avec des sommets tous différents) dans G contenant au moins k sommets ?

Ce problème est NP-Complet car, en considérant le sous-ensemble des instances où $k = |V|$, le problème correspondant est le problème NP-Complet de recherche d'un chemin hamiltonien dans un graphe [Kar72] :

HAMILTONIAN PATH

• **Entrée** : Un graphe $G = (V, E)$.

• **Sortie** : Un chemin visitant chaque sommet du graphe exactement une fois.

Le problème $k-$PATH trouve plusieurs applications pratiques en plus de son intérêt théorique, notamment en bioinformatique pour la recherche de chemins de signalisation dans un réseau d'interactions de protéines [SIKS06].

L'algorithme naïf pour répondre à ce problème consiste à énumérer l'ensemble des chemins à k sommets du graphe. Pour un graphe à n sommets, cela induit une complexité de $\mathcal{O}(n^k)$.

Il est possible de répondre au problème de recherche d'un chemin hamiltonien (donc le problème $k-$PATH pour $k = n$) en stockant dans une table de programmation dynamique tous les sous-ensembles $S \subseteq V$ pour lesquels il existe un chemin simple passant par tous les éléments de S. Comme la taille de V est n, il y a 2^n sous-ensembles S de V. La partie exponentielle de la complexité en temps et espace de l'algorithme est donc bornée par $\mathcal{O}^*(2^n)$. Ce résultat datant des années 60, il semble raisonnable de chercher à obtenir un résultat similaire (c'est-à-dire un algorithme en $\mathcal{O}^*(2^k)$) pour n'importe quel k.

Il faut attendre le milieu des années 90 et l'article de Alon, Yuster et Zwick [AYZ95] pour répondre positivement à cette attente. En utilisant la technique du color-coding en plus de la programmation dynamique précédente, ils donnent un algorithme de complexité paramétrée pour le problème $k-$PATH. Il existe une version randomisée et

une version déterministe, dont la complexité en temps est si élevée qu'elle n'est pas envisageable en pratique. Nous nous intéressons dans un premier temps à la version randomisée.

L'algorithme est composé de deux phases principales :

1. coloration aléatoire uniforme (avec la même probabilité) des sommets du graphe avec k couleurs (c'est-à-dire définition d'une fonction $col : V \to \{1, \ldots, k\}$),
2. recherche d'un chemin *colorful*, c'est-à-dire contenant chaque couleur exactement une fois, dans le graphe coloré aléatoirement par la phase 1.

Algorithme 4 : Technique du color-coding pour le problème $k-$PATH

Entrées : Un graphe $G = (V, E)$, un entier k

```
pour chaque i de 0 a e^k faire
    col tableau initialisé à vide ;
    pour chaque v ∈ V faire
        col[v] ← rand(1, k) ;
        // rand(a, b) renvoi un entier aléatoire entre a et b
    si (G, col) contient un chemin colorful à k sommets alors
        retourner Vrai ;
retourner Faux ;
```

Il est évident qu'un chemin colorful correspond à un chemin simple, ce qui est recherché. Néanmoins, il est possible qu'un chemin simple, solution au problème, ne soit pas colorful par la coloration aléatoire de la phase 1. Il va donc être nécessaire de relancer l'algorithme plusieurs fois pour rendre faible la probabilité d'erreur (c'est-à-dire ne pas trouver de chemin colorful alors qu'il existait un chemin simple).

Montrons d'abord que la recherche d'un chemin colorful dans un graphe coloré peut s'effectuer à l'aide d'un algorithme de complexité paramétrée. On résout cette phase 2 par programmation dynamique, en se basant sur le fait qu'un chemin colorful à i sommets finissant en v contient un chemin colorful à $i-1$ sommets finissant en u, un sommet voisin de v. Nous stockons $Vrai$ dans la case $D(v, S)$ de la table D de programmation dynamique s'il existe un chemin colorful utilisant toutes les couleurs de S finissant en v. Nous traduisons le fait énoncé précédemment par la récurrence suivante :

$$D(v, S) = \bigvee_{\{u,v\} \in E} D(u, S \setminus \{col(v)\}).$$

Le cas de base est le suivant :

$$D(v, \{c\}) = \begin{cases} Vrai & \text{si } c = col(v), \\ Faux & \text{sinon.} \end{cases}$$

Nous répondons Oui à la question s'il existe un v tel que $D(v, \{1, 2, \ldots, k\}) = Vrai$. La complexité de l'algorithme s'obtient en observant qu'il y a 2^k sous-ensembles de $\{1, 2, \ldots, k\}$ et i cases pour chaque sous-ensemble de taille i – il y a donc $\mathcal{O}(2^k k)$ cases. Pour chaque case, nous effectuons au maximum $|E|$ opérations. La complexité en temps de la recherche d'un chemin colorful étant donné un graphe coloré est donc de $\mathcal{O}(2^k k|E|)$.

Remarquons que seules les couleurs correspondant aux chemins sont stockés dans la table et non l'ensemble des sommets de ceux-ci. Ainsi, l'explosion combinatoire de l'algorithme est confiné au nombre k de couleurs et non au nombre n de sommets.

Finalement, l'algorithme recherchant un chemin simple à k sommets dans un graphe non-coloré est également de complexité paramétrée. En effet, nous avons vu que la phase 2 peut s'effectuer en temps $\mathcal{O}^*(2^k)$. Nous observons maintenant qu'en relançant un nombre exponentiel en k seulement la phase 1 de coloration, nous obtenons une probabilité d'erreur faible. Pour cela, nous remarquons que la probabilité qu'un chemin simple à k sommets soit colorful est de $\frac{\text{\# colorations colorful}}{\text{\# colorations possibles}} = \frac{k!}{k^k} > \frac{1}{e^k}$ (par l'approximation de Stirling). En effet, il y a k^k façons différentes de colorier un chemin à k sommets avec k couleurs différentes, mais seulement $k!$ manières d'obtenir un chemin colorful à k sommets – une couleur utilisée n'est plus disponible pour les autres sommets du chemin. Nous concluons sur la complexité globale de l'algorithme randomisé en observant qu'il faut effectuer e^k essais de colorations et qu'un essai s'effectue en $\mathcal{O}(2^k k|E|)$: la complexité globale de l'algorithme est donc de $\mathcal{O}(2e^k k|E|) = \mathcal{O}(5.44^k k|E|)$. L'idée de l'algorithme est donnée dans l'Algorithme 4. Notons que ce squelette est commun à tout algorithme utilisant la technique du color-coding.

Il existe également une version déterministe de l'algorithme proposé par Alon *et al.*, basée sur les *familles de fonctions de hachage k-parfaites*. Malheureusement, l'utilisation de ces familles mène à un algorithme de complexité $\mathcal{O}^*(c^k)$, pour un c très grand rendant improbable leur utilisation en pratique ($c > 8000$).

De manière assez surprenante, augmenter le nombre de couleurs pour obtenir un chemin simple à k sommets diminue la complexité pire cas de l'algorithme randomisé. En augmentant le nombre de couleurs, nous cédons au compromis suivant : moins d'essais sont nécessaires car un chemin à plus de chances d'être colorful, cependant, chaque essai prend plus de temps. C'est le choix effectué par Hüffner *et al.* [HWZ08], prouvant que l'utilisation de $1.3k$ couleurs mène à un algorithme pour le problème $k-$PATH de complexité $\mathcal{O}^*(4.32^k)$. Cependant, augmenter le nombre de couleurs mène à une plus grande complexité en espace, qui est souvent la limite pratique des algorithmes basés sur la programmation dynamique.

Enfin, deux groupes ont proposé indépendamment une méthode pour résoudre le problème $k-$PATH utilisant la technique du color-coding avec la technique du *divide-and-conquer* (diviser pour régner) [KMRR06, CLSZ07]. Cette nouvelle technique, logiquement appelée *divide-and-color*, permet d'obtenir un algorithme randomisé de complexité

$\mathcal{O}^*(4^k)$. Ils améliorent aussi grandement la complexité pour l'algorithme déterministe, obtenant une complexité de $\mathcal{O}^*(16^k)$ pour [KMRR06] et $\mathcal{O}^*(12.8^k)$ pour [CLSZ07].

1.3.2.2 Détection de monômes multilinéaires

La technique de la détection de monômes multilinéaires fut introduite récemment par Koutis et Williams [Kou08, KW09]. Elle permet d'obtenir un algorithme randomisé de complexité $\mathcal{O}^*(2^k)$ en temps et polynomial en espace pour répondre au problème $k-$PATH. Intuitivement, l'idée est d'encoder le graphe dans un *circuit arithmétique* contenant une version compressée d'un polynôme à plusieurs variables, puis de chercher, dans ce polynôme, un monôme multilinéaire de taille k, c'est-à-dire contenant exactement k variables de degré 1.

Plus formellement, notons X un ensemble de variables $\{x_1, x_2, \ldots\}$. Un polynôme à plusieurs variables (multivarié) est une somme de monômes. Le degré d'un monôme est la somme des degrés des variables le composant. Par exemple, le monôme $m_1 = x_1x_2^2$ est de degré $1 + 2 = 3$. Un monôme est dit *multilinéaire* si le degré de chacune de ses variables est égal à 1. Ainsi, un monôme multilinéaire de degré k contient k variables différentes. Le monôme m_1 pris en exemple ci-dessus n'est pas multilinéaire car la variable x_2 est au carrée. Le monôme $m_2 = x_1x_2x_3$, en revanche, est de degré 3 et multilinéaire.

Un *circuit arithmétique* A est un DAG, tel que chaque feuille (sommet sans arc sortant) contient une variable de X et chaque sommet interne contient une opération $+$ ou $\times$. Il est possible de représenter un polynôme à l'aide d'un tel circuit arithmétique, comme montré en exemple Figure 1.10. Le polynôme correspondant à une feuille est le polynôme contenant la variable de la feuille. Le polynôme d'un sommet interne contenant l'opérateur $+$ (resp. $\times$) correspond à la somme (resp. au produit) des polynômes de ses fils. Le circuit A représente le polynôme obtenu depuis la *racine* de A, un sommet distinct du circuit.

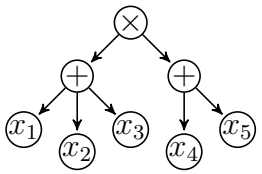


FIGURE 1.10 – Le circuit arithmétique représentant le polynôme $x_1x_4 + x_1x_5 + x_2x_4 + x_2x_5 + x_3x_4 + x_3x_5 = (x_1 + x_2 + x_3)(x_4 + x_5)$.

Le problème suivant est fondamental pour le fonctionnement de la technique :

MULTILINEAR DETECTION

• **Entrée** : Un circuit arithmétique A représentant un polynôme P sur un ensemble X de variables, un entier k.

• **Sortie** : Est-ce que P contient un monôme multilinéaire de degré k ?

Théorème 1.3.1. *([Wil09, KW09]) Il est possible de résoudre le problème* MULTILINEAR DETECTION *par un algorithme randomisé de complexité en temps $\tilde{\mathcal{O}}(2^k|A|)$ et de complexité en espace $\tilde{\mathcal{O}}(|A|)$.*

Précisons que la notation $\tilde{\mathcal{O}}$ supprime les facteurs polylogarithmiques. Ce théorème permet l'obtention d'un algorithme pour le problème MULTILINEAR DETECTION où la taille du polynôme n'intervient pas dans la partie exponentielle de la complexité. Il reste à montrer comment encoder le graphe en un circuit arithmétique pour utiliser ce résultat pour le problème $k-$PATH.

Une variable par sommet du graphe G de taille n est créée – l'ensemble des variables utilisées est donc $X = \{x_1, x_2, \ldots, x_n\}$.

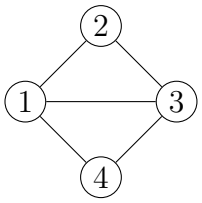
Nous construisons ensuite, par programmation dynamique, un circuit arithmétique représentant un polynôme. Le circuit est construit tel que tous les chemins (non forcement simples) à k sommets du graphe G sont encodés par un monôme du polynôme. L'encodage d'un chemin à k sommets est le produit des variables qui correspondent à ses sommets. Par exemple, le chemin $(1, 3, 5, 1, 2)$ est encodé par le monôme $x_1x_3x_5x_1x_2 = x_1^2x_2x_3x_5$.

La construction récursive suivante permet, de manière assez similaire à la construction par programmation dynamique utilisant la technique du color-coding vue précédemment, de construire les monômes de degré k qui correspondent à l'ensemble des chemins à k sommets de G. Intuitivement, le *OU* logique de la construction précédente est remplacée par une somme. En effet, il existe une solution dans le polynôme si et seulement si il existe une solution dans un des monômes le constituant. Une illustration de cet encodage est donnée Figure 1.11.

$$\begin{aligned} P(1, v) &= x_v, \\ P(i, v) &= \sum_{\{u,v\}\in E} x_v P(i-1, u), \\ Q(k) &= \sum_{v\in V} P(k, v). \end{aligned}$$

Le polynôme $P(i, v)$ est une somme de l'ensemble des monômes correspondant à un chemin à i sommets finissant en v. De tels monômes correspondent à un produit entre x_v et un monôme correspondant à un chemin à $i-1$ sommets finissant en u, un sommet voisin de v (de tels monômes sont stockés dans $P(i-1, u)$).

Chaque monôme dans le polynôme $Q(k)$ correspond à un chemin à k sommets dans G. Il est possible de montrer par induction qu'un monôme multilinéaire de degré k dans $Q(k)$ correspond à un chemin simple à k sommets dans G. En effet, une variable apparaît plusieurs fois dans le monôme si et seulement si un sommet apparaît plusieurs fois dans le chemin correspondant. En utilisant le Théorème 1.3.1, nous déduisons un algorithme randomisé de complexité $\mathcal{O}^*(2^k)$ pour le problème $k-$PATH.



$$
\begin{aligned}
Q(k) = \;& Q(3) = P(3,1) + P(3,2) + P(3,3) + P(3,4) \\
P(3,1) = \;& x_1P(2,2) + x_1P(2,3) + x_1P(2,4) \\
=\;& x_1(x_2P(1,1) + x_2P(1,3))+ \\
& x_1(x_3P(1,1) + x_3P(1,2) + x_3P(1,4))+ \\
& x_1(x_4P(1,1) + x_4P(1,3)) \\
=\;& x_1x_2x_1 + x_1x_2x_3+ \\
& x_1x_3x_1 + x_1x_3x_2 + x_1x_3x_4+ \\
& x_1x_4x_1 + x_1x_4x_3
\end{aligned}
$$

FIGURE 1.11 – Illustration de la technique de détection de monômes multilinéaires pour la recherche d'un chemin simple à 3 sommets dans un graphe à 4 sommets. Nous n'avons développé que le polynôme $P(3,1)$, où chaque monôme correspond à un chemin (non forcement simple) à 3 sommets, finissant sur le sommet 1.

1.3.2.3 Autres techniques

Outre les techniques vues précédemment, un certain nombre de techniques génériques ont été mises au point pour prouver qu'un problème paramétré appartient à la classe FPT. Nous présentons brièvement dans la suite l'utilisation d'*arbres de recherche bornés*, la *compression itérative* et la *localisation gloutonne*. D'autres techniques et plus d'informations se trouvent dans la partie II de [Nie06].

L'idée dans l'utilisation d'*arbres de recherche bornés* est d'effectuer une recherche systématique exhaustive que l'on organise selon un arbre. On obtient un algorithme de complexité paramétrée en bornant la hauteur de cet arbre par une fonction dépendant uniquement du paramètre. Prenons pour exemple le problème VERTEX COVER où il existe la propriété suivante : pour chaque arête $\{u, v\} \in E$, soit u, soit v, soit les deux sont dans une solution. On construit l'arbre binaire complet comme présenté Figure 1.12. Pour chaque sommet interne, on effectue le traitement polynomial consistant à prendre une arête $\{u, v\}$ dans le graphe courant et pour chacun des deux fils, enlever le sommet au graphe que l'on ajoute à la solution courante (associée au sommet). On stoppe ce processus lorsque k arêtes ont été ajoutées à la solution – l'arbre est de hauteur k. Pour chaque feuille, s'il reste des arêtes dans le graphe, alors l'ensemble de sommets associé à la feuille n'est pas solution au problème, cet ensemble ne couvre pas l'ensemble des arêtes du graphe. Comme il y a $2^{k+1} - 1$ sommets dans l'arbre et que le traitement est

polynomial pour chaque sommet, l'algorithme exhaustif visitant chaque sommet de l'arbre de recherche est donc exponentiel seulement selon k, la taille de la solution. On trouve également toutes les solutions de taille k dans les feuilles de l'arbre.

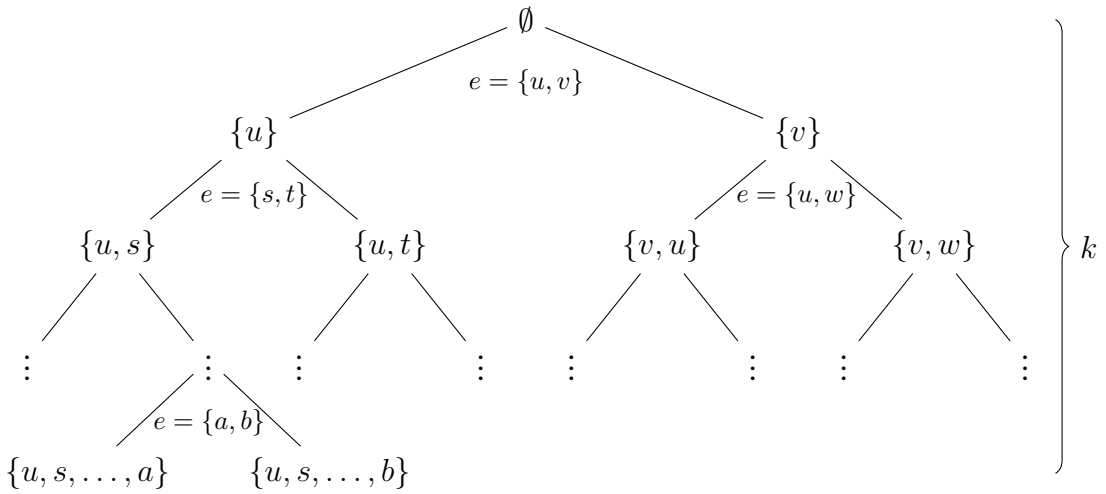


FIGURE 1.12 – Arbre de recherche borné pour répondre au problème VERTEX COVER. Chaque sommet est étiqueté par la valeur courante du vertex cover.

La méthode de la *compression itérative* (*iterative compression*) a été introduite en 2004 [RSV04]. Cette technique s'applique pour les problèmes de minimisation, paramétrés par la taille k de leurs solutions – elle a été utilisée depuis pour de nombreux problèmes nécessitant des suppressions de sommets. L'idée est de construire une solution pour une instance de taille $i+1$ à partir d'une solution pour une instance de taille i. Ainsi, si S est une solution de taille k pour une instance G de taille i (sans perte de généralité, G contient les sommets $\{v_1, \ldots, v_i\}$), alors $S \cup \{v_{i+1}\}$ est une solution de taille $k+1$ pour une instance $G \cup \{v_{i+1}\}$ de taille $i+1$. Reste ensuite à effectuer la phase délicate de *compression*, avec un algorithme de complexité paramétrée, permettant :

- soit d'obtenir une solution de taille k pour cette nouvelle instance de taille $i+1$,
- soit de prouver qu'une telle solution de taille k n'existe pas.

Un exemple de cette technique sur le problème VERTEX COVER se trouve dans le Chapitre 11 de [Nie06].

A l'inverse de la compression itérative, la *localisation gloutonne* (*greedy localization*) s'utilise sur des problèmes de maximisation, également paramétrés par la taille de la solution. L'idée est de construire de manière gloutonne la plus grande solution possible. Par la suite,

- soit la solution calculée est de taille satisfaisante (c'est-à-dire supérieure au paramètre) et on répond au problème,
- soit on utilise cette solution pour en construire une plus grande, si elle existe.

Enfin, nous verrons dans la Section 1.3.3 que la recherche de noyau (*kernelization*) permet également de prouver qu'un problème est dans la classe FPT. Cette technique

est devenue ces dernières années un domaine de recherche à part entière, avec une théorie associée. C'est pourquoi nous lui dédions une section entière.

1.3.2.4 W[1]-Difficulté

Nous avons vu précédemment que, comme pour la classe P, il suffit d'exhiber un algorithme de complexité paramétrée pour montrer qu'un problème appartient à la classe FPT. Comme pour la complexité classique, il est possible de prouver qu'un algorithme n'appartient pas à la classe FPT à l'aide de réductions.

Toutefois, il est parfois simple de montrer qu'un problème n'est pas dans FPT. En effet, un problème paramétré par k n'est, par définition, pas dans la classe FPT si on ne peut pas le résoudre en temps polynomial pour chaque k fixé. Prenons pour exemple le problème suivant :

GRAPH COLORING

• **Entrée** : Un graphe $G = (V, E)$, un entier k.

• **Sortie** : Une coloration des sommets de G à l'aide de k couleurs, telle que pour chaque arête $\{u, v\} \in E$, la couleur de u est différente de la couleur de v, c'est-à-dire une partition de V en k ensembles disjoints $V_1, V_2, \ldots, V_k$ tels que chaque V_i est indépendant.

Il est connu que le problème GRAPH COLORING est NP-Complet pour $k = 3$; on ne peut pas trouver en temps polynomial une coloration du graphe avec seulement trois couleurs, sauf si P = NP. Ce problème ne peut pas être dans la classe FPT, avec comme paramètre le nombre k de couleurs. En effet, un algorithme de complexité paramétrée pour une entrée de taille n serait de la forme $f(k).n^c$, c une constante, et donnerait un algorithme polynomial pour $k = 3$ (on note néanmoins que le problème dual est dans FPT, c'est-à-dire lorsqu'il a pour paramètre $|V| - k$). Cependant, cette condition n'est pas suffisante. Par exemple, on peut résoudre de manière polynomiale le problème INDEPENDENT SET pour chaque k fixé, mais il est prouvé que ce problème n'est pas dans FPT. Nous voyons par la suite comment obtenir une telle preuve.

Nous avons déjà vu précédemment le problème SAT (aussi nommé CNF-SATISFIABILITY), qui est central dans la théorie de la complexité classique. Une version paramétrée par le poids de ce problème est donnée ci-après, où le *poids d'une affectation* est défini comme le nombre de variables mises à la valeur *Vrai*.

WEIGHTED CNF-SATISFIABILITY

• **Entrée** : Un ensemble X de n variables booléennes, une formule F en forme normale conjonctive (CNF), c'est-à-dire une conjonction (ET logique) de clauses, où les clauses sont des disjonctions (OU logique) de littéraux (un littéral est une variable de X ou sa négation) et un entier k.

• **Sortie** : Une affectation des variables de X rendant vraie la formule F et de poids exactement k.

Le problème appelé WEIGHTED 2-CNF-SATISFIABILITY est équivalent, mais chaque clause de C contient au plus deux littéraux. On sait que le problème 2-SAT est dans la classe P, alors que le problème 3-SAT est NP-Complet [GJ79]. De manière assez surprenante, on ne connait pas d'algorithme de complexité paramétrée pour le problème WEIGHTED 2-CNF-SATISFIABILITY. Pour continuer l'analogie, trouver un algorithme polynomial pour le problème 3-SAT est peu probable, tout comme trouver un algorithme de complexité paramétrée pour le problème WEIGHTED 2-CNF-SATISFIABILITY. Ce dernier joue donc un rôle central dans la théorie de la complexité paramétrée.

La notion de réduction polynomiale définie Section 1.2.2 n'est pas suffisante pour la théorie de la complexité paramétrée. On cherche cette fois à préserver le paramètre au cours de la réduction, à contrôler sa taille. C'est pourquoi la FPT-réduction a été introduite. On construit une FPT-réduction d'un problème paramétré P_1 vers un problème paramétré P_2 si pour chaque instance (X_1, k_1) de P_1, on construit une instance (X_2, k_2) de P_2 telle que :

1. (X_1, k_1) est vraie pour P_1 si et seulement si (X_2, k_2) est vraie pour P_2,
2. $k_2 = g(k_1)$ pour une fonction g quelconque ne dépendant que de k_1,
3. la construction de (I_2, k_2) est faite en temps $f(k_1).|I_1|^c$, pour une fonction f quelconque ne dépendant que de k_1 et c une constante.

Il s'avère que borner le paramètre est souvent la principale difficulté lors de l'élaboration d'une telle réduction.

Comme pour la complexité classique, deux problèmes proches selon leur définition peuvent ne pas appartenir à la même classe de complexité. Par exemple, les problèmes INDEPENDENT SET et VERTEX COVER sont très proches – étant donnée une solution S de taille k pour le problème VERTEX COVER, $V \setminus S$ est une solution pour le problème INDEPENDENT SET de paramètre $|V| - k$ (voir Figure 1.13).

Pourtant, si le problème VERTEX COVER peut se résoudre avec un algorithme de complexité paramétrée, ce n'est pas le cas pour le problème INDEPENDENT SET. La propriété vue précédemment entre les deux solutions ne préserve pas le paramètre (le second point de la FPT-réduction), ce dernier dépend également du nombre de sommets dans le graphe. En revanche, le dual du problème INDEPENDENT SET (où le paramètre est $k' = |V| - k$) est donc dans FPT par cette même propriété, en utilisant l'algorithme de

complexité paramétrée pour le problème VERTEX COVER. Plus généralement, il arrive souvent qu'un problème et son dual ne soient pas dans la même classe de complexité.

$a)$ $b)$

FIGURE 1.13 – Étant donné un même graphe, une solution optimale coloriée en noir pour le problème VERTEX COVER est montrée en $a)$ et pour le problème INDEPENDENT SET en $b)$.

La FPT-réduction est transitive, comme la réduction polynomiale. Comme pour la complexité classique, des classes de complexités ont été introduites. Si le problème SAT est central pour la complexité classique, les classes de complexité paramétrée sont définies autour des problèmes WEIGHTED CNF-SATISFIABILITY. Ainsi, muni de réductions appropriées et de classes de complexité, on pourra indiquer que si un des problèmes de ces classes accepte un algorithme de complexité paramétrée, alors tous les problèmes de ces classes en acceptent un.

On peut alors définir ceci :

- on appelle W[1] la classe contenant tous les problèmes pouvant être réduits au problème WEIGHTED 2-CNF-SATISFIABILITY par une FPT-réduction,
- un problème est W[1]-Difficile si le problème WEIGHTED 2-CNF-SATISFIABILITY peut être réduit à lui par une FPT-réduction,
- comme pour la complexité classique, un problème est W[1]-Complet s'il est dans W[1] et W[1]-Difficile.

La classe W[2] est définie de manière analogue à W[1] en remplaçant le problème WEIGHTED 2-CNF-SATISFIABILITY par le problème WEIGHTED CNF-SATISFIABILITY.

On peut, en fait, aller plus loin dans la définition de ces classes. On dit qu'une formule booléenne est t-normale si c'est une *conjonction-de-disjonctions-de-conjonctions-...* de littéraux, avec $t-1$ alternances entre conjonctions et disjonctions. Par exemple, une formule 2-CNF est juste une conjonction de clauses de taille 2, elle est 1-normale, tandis qu'une formule CNF est une conjonction de disjonctions (la taille des clauses n'est pas bornée), elle est 2-normale.

Le problème WEIGHTED t-NORMALIZED SATISFIABILITY est défini de manière analogue au problème WEIGHTED CNF-SATISFIABILITY, mais la formule CNF d'entrée est t-normale.

Sont alors définies les classes W[t], $t \geq 1$, telles que W[t] est la classe où se trouvent tous les problèmes paramétrés pouvant être réduits au problème WEIGHTED t-NORMALIZED SATISFIABILITY par une FPT-réduction.

Généralement seules les deux classes de plus bas niveaux (W[1] et W[2]) nous intéresserons – la plupart des problèmes naturels difficiles sont soit W[1]-Difficiles, soit

W[2]-Difficiles.

On peut facilement montrer que le problème INDEPENDENT SET appartient à la classe W[1]. Un graphe $G = (V, E)$ a une solution pour le problème INDEPENDENT SET de taille k si et seulement si la formule 2-CNF suivante :

$$\bigwedge_{\{u,v\}\in E} (\overline{x_u} \vee \overline{x_v}),$$

a une solution de poids exactement k. En effet, si une clause n'est pas vraie, alors c'est que les deux variables sont vraies, indiquant que deux sommets voisins sont dans la solution pour le problème INDEPENDENT SET, ce qui est impossible.

Il est prouvé que le problème WEIGHTED ANTIMONOTONE 2-CNF-SATISFIABILITY, équivalent au problème WEIGHTED 2-CNF-SATISFIABILITY mais où tous les littéraux sont négatifs, est W[1]-Complet. Alors, la réduction inverse est valide et prouve que le problème INDEPENDENT SET est W[1]-Complet.

Avec une construction similaire, la formule 2-CNF suivante :

$$\bigwedge_{\{u,v\}\in E} (x_u \vee x_v),$$

indique que le problème VERTEX COVER est dans W[1]. En fait, ceci est peu informatif car FPT $\subseteq$ W[1], par l'utilisation de FPT-réduction dans la définition de W[1].

Plus largement, on a les inclusions suivantes (qui sont conjecturées strictes) :

$$\mathsf{P} \subseteq \mathsf{FPT} \subseteq \mathsf{W}[1] \subseteq \mathsf{W}[2] \subseteq \cdots \subseteq \mathsf{W}[t] \subseteq \mathsf{XP}$$

La classe XP correspond aux problèmes que l'on peut résoudre en $\mathcal{O}(n^{g(k)})$, où g est une fonction quelconque – donc en temps polynomial pour chaque k fixé. Par exemple, le problème GRAPH COLORING n'est pas dans XP (sauf si P = NP) car il est NP-Complet même pour $k = 3$. On sait que FPT $\subsetneq$ XP.

La conjecture analogue à P $\neq$ NP est que FPT $\neq$ W[1]. Ainsi, un problème W[1]-Complet appartient aux problèmes les plus difficiles de la classe W[1] – il n'est pas dans la classe FPT (si la conjecture est vraie).

C'est de leur étonnement de ne pas trouver d'algorithme de complexité paramétrée pour le problème INDEPENDENT SET, semblant pourtant très proche du problème VERTEX COVER, qu'est née la volonté de Downey et Fellows de créer une théorie de la complexité paramétrée [DF99].

1.3.3 Recherche de noyaux

Effectuer un pré-traitement sur les données à traiter pour diminuer le temps de calcul est une technique certainement aussi ancienne que la conception d'algorithmes. Aujourd'hui encore, des logiciels commerciaux comme CPLEX utilisent des pré-traitements pour résoudre des programmes linéaires. Sans ces pré-traitements, certains de ces programmes linéaires seraient intraitables [Bix02]. L'idée centrale est la suivante : couper en temps polynomial les parties simples à résoudre pour ne garder que le noyau (*kernel*) du problème, la partie réellement difficile. De manière informelle, l'obtention d'un noyau s'effectue par une succession de règles de réduction, telle que l'instance obtenue après l'application de ces règles soit vraie si et seulement si l'instance de départ est vraie. Ainsi, un pré-traitement polynomial pour un problème NP-Complet permet de réduire la taille des données traitées par un algorithme de complexité exponentielle.

Même si la description précédente semble uniquement apporter un gain en pratique, il est possible d'obtenir des résultats théoriques, en lien avec la complexité paramétrée. En effet, le but de la recherche de noyau est d'obtenir en temps polynomial une instance équivalente (le noyau) dont on peut borner la taille. Dans le cadre de la complexité paramétrée, on cherche à borner cette taille par une fonction dépendant uniquement du paramètre k. On pourra par la suite résoudre de manière optimale cette instance réduite.

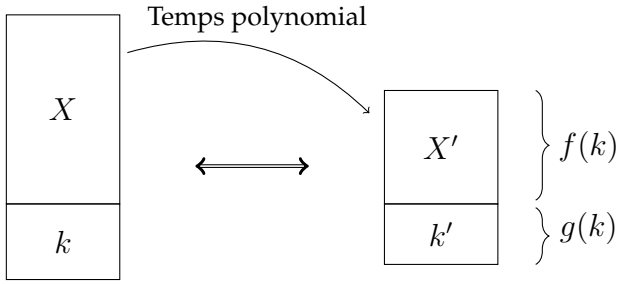


FIGURE 1.14 – La kernalization : depuis une instance (X, k), on obtient en temps polynomial une instance équivalente (X', k') dont la taille est bornée par une fonction de k.

Plus formellement, étant donnée une instance (X, k) d'un problème, avec X l'entrée et k le paramètre, obtenir un noyau consiste à obtenir, en temps polynomial, une instance (X', k') telle que :

- $k' \leq g(k)$,
- $|X'| \leq f(k)$,
- (X, k) est vrai si et seulement si (X', k') est vrai.

Voir aussi Figure 1.14. La fonction g ne doit dépendre uniquement de k, cependant, dans les faits, nous avons souvent $k' \leq k$ (Niedermeier indique ne pas connaître de problème naturel ayant un noyau où $k' > k$ [Nie06]). La fonction f donne la taille du noyau et ne

doit dépendre que de k. C'est borner cette taille qui est crucial. Le Chapitre 7 du livre de Niedermeier est consacré aux noyaux [Nie06].

La recherche de noyau est une technique assez utilisée. En effet, on sait que l'existence d'un noyau implique que le problème est dans FPT (et inversement) [Nie06]. Le sens direct est assez simple – utiliser une algorithme exhaustif sur l'instance du noyau est de complexité paramétrée car la taille de cette instance est dépendante de k uniquement. Ainsi, n'importe quel algorithme appliqué à cette instance aura une complexité dépendante uniquement de k. Pour le sens retour, deux cas sont possibles selon la valeur de $n = |X|$. On suppose qu'un algorithme de complexité paramétrée est connu, de complexité $f(k).n^c$, avec c une constance. Alors, si $n \geq f(k)$, la complexité de l'algorithme est bornée par n^{c+1} et renvoie donc en temps polynomial une instance triviale *OUI* ou *NON*. Si $n < f(k)$, alors la taille de l'instance est bornée par $f(k)$, c'est un noyau de taille $f(k)$.

Deux remarques sont à donner sur cette propriété. Premièrement, le sens retour est non constructif – on ne peut pas donner la construction d'un noyau à partir d'un algorithme de complexité paramétrée. D'autre part, la complexité de l'algorithme de complexité paramétrée obtenu depuis le noyau peut être assez grande car la taille de ce dernier peut être exponentielle en k. C'est pourquoi beaucoup de recherches récentes visent à montrer l'existence de noyaux ayant une taille polynomiale selon k.

Afin de donner un exemple sur l'obtention d'un noyau, on expose la méthode suivante pour obtenir un noyau polynomial pour le problème VERTEX COVER, due à une communication privée de Sam Buss (répertoriée dans [DF99]) et considérée maintenant comme appartenant au folklore. Elle est basée sur les trois règles suivantes appliquées successivement et tant que possible sur l'instance $(G = (V, E), k)$ du problème VERTEX COVER :

1. si un sommet $u \in V$ est isolé (c'est-à-dire de degré 0), alors supprimer u et laisser k inchangé,
2. si un sommet $u \in V$ est de degré 1 et $\{u, v\} \in E$, alors supprimer v du graphe (avec toutes ses arêtes incidentes) et diminuer k de 1,
3. si un sommet $u \in V$ a un degré strictement supérieur à k, alors supprimer u (avec toutes ses arêtes incidentes) et diminuer k de 1.

La première règle est trivialement correcte.

Pour la seconde règle, considérons une solution S contenant u. Alors, $S' = (S \setminus \{u\}) \cup \{v\}$ est aussi une solution car l'arête $\{u, v\}$ est toujours couverte et toutes les autres arêtes sont toujours couvertes.

Enfin, pour la troisième règle, supposons qu'il existe une solution S ne contenant pas u. Alors, pour être une solution, S doit contenir l'ensemble des voisins de u afin de couvrir ces arêtes. Comme u est de degré strictement supérieur à k, il existe au moins $k + 1$ arêtes à couvrir. Alors, S doit contenir au moins ces $k + 1$ sommets, ce qui est une contradiction avec la taille k de la solution.

Comme ces règles s'appliquent clairement en temps polynomial, il reste à montrer que la taille de l'instance obtenue après l'application de ces règles est une fonction de k (ici la fonction est polynomiale). Si, au cours de l'application des règles on obtient $k < 0$, alors on peut répondre Non au problème – il n'existe pas de solution de taille k couvrant les arêtes du graphe. On observe qu'après l'application exhaustive des règles, aucun sommet n'a de degré supérieur à k – chaque sommet dans la solution peut ne couvrir que k arêtes au maximum. Comme on suppose qu'il existe une solution de taille k, alors il ne peut donc avoir plus de k^2 arêtes dans le graphe (on répond Non s'il y a plus de k^2 arêtes). En effet, chacun des k sommets de la solution ne peut être voisin que d'au plus k sommets. Enfin, on sait par la première règle que chaque sommet est incident à au moins une arête, il y a donc au plus k^2 sommets dans le graphe.

On note qu'il a été depuis montré qu'un noyau de taille linéaire ($2k$) était possible pour le problème VERTEX COVER (on trouve dans les notes bibliographiques du Chapitre 7 de [Nie06] les références vers les multiples résultats menant à ce noyau de taille linéaire). Cependant, l'algorithme que nous avons repris précédemment illustre la méthodologie habituelle pour l'obtention d'un noyau : une série de règles de réductions sûres donnant une instance équivalente de taille plus petite, puis une analyse bornant la taille de l'instance positive restante (si la taille est supérieure à cette borne, c'est une instance négative).

Comme pour prouver qu'un algorithme n'est pas polynomial ou de complexité paramétrée, il est également possible de montrer que l'obtention d'un noyau polynomial est improbable. L'article de revue récent de Bodlaender donne un aperçu des récentes techniques pour prouver ces bornes inférieures pour des noyaux [Bod09].

Prenons à nouveau pour exemple le problème $k-$PATH, que nous avons vu dans la classe FPT précédemment. Le problème admet donc un noyau, mais celui-ci peut-il être de taille polynomiale selon k ? Récemment, Bodlaender *et al.* ont répondu par la négative [BDFH09]. On essaie ci-dessous d'en donner l'intuition.

Étant données t instances du problème, $(G_1, k), (G_2, k), \ldots, (G_t, k)$ avec le même paramètre k, on construit une nouvelle instance (G, k') telle que le graphe G est une union disjointe des graphes $G_i, 1 \leq i \leq t$, et on pose $k' = k$ (voir Figure 1.15). Le graphe G est donc une union de t composantes connexes. On a la propriété suivante immédiate : il existe un chemin simple à k sommets dans G si et seulement si il existe un chemin simple à k sommets dans un des graphes G_i. S'il existait un noyau polynomial pour ce problème, alors on obtiendrait une instance (G', k'') telle que $|G'| = k'^c$, avec c une constante. Potentiellement, ce nombre est plus petit que t, par exemple si $t = k^{2c}$. Cela voudrait donc dire qu'il serait possible, en temps polynomial, d'élaguer des composantes connexes de G pour obtenir G' et donc indiquer des instances $G_i, 1 \leq i \leq t$, ne contenant pas de chemin simple à k sommets. Ceci semble, intuitivement, au moins aussi difficile que de résoudre le problème $k-$PATH lui-même – problème qui est NP-Complet.

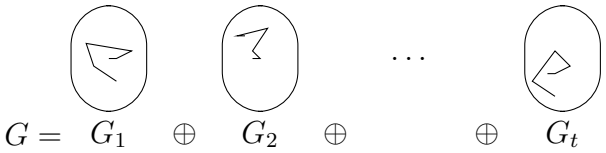


FIGURE 1.15 – Illustration de la construction d'une instance (G, k) pour le problème $k-$PATH, étant données t instances (G_i, k) du problème. Le graphe G est construit comme étant l'union disjointe des t graphes. Comme l'union est disjointe, on voit clairement qu'il existe un chemin simple à k sommets dans G si et seulement si il existe un chemin simple à k sommets dans un des t graphes.

Ceci peut se prouver plus formellement. Un algorithme est dit de *Or-Composition* pour un problème paramétré si :

- il reçoit une séquence $((x_1, k), (x_2, k), \ldots, (x_t, k))$ d'instances du problème,
- a une complexité polynomiale en $\sum_{i=1}^{t} |x_i| + k$,
- retourne une instance (y, k') du même problème telle que :
  - (y, k') est vrai si et seulement si *il existe un* i tel que (x_i, k) est vrai,
  - k' est polynomial en k.

Sans entrer dans les détails, il a été montré que l'existence d'un noyau polynomial pour un problème admettant une algorithme de *Or-Composition* est peu plausible. La construction par union disjointe vue précédemment pour le problème $k-$PATH respecte les conditions de l'*Or-Composition*.

1.3.4 Algorithmes d'approximation

Lorsqu'un problème d'optimisation est difficile, on sait qu'il est impossible de trouver un algorithme polynomial (sauf si P = NP) donnant une solution *optimale*. Il est assez naturel de se demander si relâcher *de manière contrôlée* la contrainte d'optimalité pourrait permettre d'obtenir une complexité polynomiale. En d'autres termes, on aimerait un algorithme rapide, retournant une solution non optimale mais dont on peut borner l'erreur.

Même si leur but est de contourner la difficulté des problèmes NP-Complets, les premiers algorithmes d'approximation ont été conçus dans les années 1960, avant la théorie de la NP-Complétude [Vaz07].

Plus formellement, si P est un problème d'optimisation (c'est-à-dire de minimisation ou de maximisation), il existe pour chaque instance I de P un ensemble de solutions faisables $F(I)$. Chaque solution $s \in F(I)$ a un certain coût $c(s)$, assumé positif (par exemple le nombre de kilomètres pour le problème MINIMUM TRAVELLING SALESMAN

PROBLEM ou le nombre de sommets retournés par une solution du problème MINIMUM VERTEX COVER). On appelle $OPT(I)$ la solution optimale pour I telle que $OPT(I) = \min_{s \in F(I)} c(s)$ si P est un problème de minimisation, et $OPT(I) = \max_{s \in F(I)} c(s)$ si P est un problème de maximisation. Soit A un algorithme qui pour n'importe quelle instance I de P retourne une solution faisable $A(I) \in F(I)$. On dira alors que A est un algorithme de *r-approximation* (où $r > 1$ est le *ratio*), si pour chaque instance I :

– A s'exécute en temps polynomial,

– $\begin{cases} OPT(I) \leq A(I) \leq r \times OPT(I) & \text{si } P \text{ est un problème de minimisation,} \\ \frac{OPT(I)}{r} \leq A(I) \leq OPT(I) & \text{si } P \text{ est un problème de maximisation.} \end{cases}$

Ainsi, le ratio r donne la borne d'erreur de l'algorithme d'approximation par rapport à la solution optimale. Par exemple un algorithme de 2-approximation a un ratio 2 et retournera toujours une solution au pire deux fois plus grande (ou plus petite) que la solution optimale. Un algorithme de 2-approximation pour le problème MINIMUM TRAVELLING SALESMAN PROBLEM (problème de minimisation) retournerait au pire une distance totale de 200 kilomètres si la solution optimale était de 100 kilomètres. On remarque qu'un algorithme de 1-approximation est exact mais non probable pour un problème NP-Complet car devant être de complexité polynomiale.

Différents types de ratios peuvent être exhibés.

Nous avons discuté précédemment d'une approximation à ratio *constant* : pour n'importe quelle instance d'entrée, l'algorithme donne en temps polynomial une solution éloignée à un facteur constant de l'optimum. Ces problèmes sont dans la classe de complexité APX.

Certains problèmes sont plus difficiles à approximer – le meilleur ratio croît en fonction de la taille de l'instance. Plus l'instance est grande, plus on est éloigné de la solution optimale. C'est par exemple le cas du problème INDEPENDENT SET que l'on sait approximer à ratio $\mathcal{O}(\frac{|V|}{(\log |V|)^2})$.

Enfin, certains algorithmes peuvent donner une solution aussi proche de l'optimum que l'on veut au prix d'un temps de calcul de plus en plus grand. On parle alors de *schéma d'approximation* qui, pour une instance donnée, prend également un ϵ quelconque, tel que le ratio de l'algorithme est de $(1+\epsilon)$. On parle de *schéma d'approximation polynomial* si pour un $\epsilon > 0$ fixé, l'algorithme est polynomial par rapport à la taille de l'instance – il appartient alors à la classe PTAS (pour Polynomial-Time Approximation Scheme). Par définition, la complexité de l'algorithme peut alors dépendre exponentiellement de $\frac{1}{\epsilon}$. Un exemple de complexité d'un tel algorithme serait $\mathcal{O}(n^{\frac{1}{\epsilon}})$ ou $\mathcal{O}(2^{\frac{1}{\epsilon}} n^2)$. Ainsi, plus ϵ est petit, meilleure est la solution obtenue – au prix d'une complexité en temps exponentiellement pire.

Contrairement aux schémas d'approximations polynomiaux, les *schémas d'approximation entièrement polynomiaux*, ont une complexité en temps polynomiale en la taille de l'instance mais aussi en $\frac{1}{\epsilon}$ – par exemple $\mathcal{O}(n^2(\frac{1}{\epsilon})^3)$. Ces problèmes sont dans la classe

FPTAS (pour Fully Polynomial-Time Approximation Scheme) et sont, techniquement, ce qu'on peut espérer de mieux pour un problème NP-Complet.

Cependant, de tels algorithmes sont très rares car les conditions sont très contraignantes. C'est pourquoi il a été introduit les *schémas d'approximations polynomiaux efficaces*. Ces problèmes ont une complexité de type $f(\epsilon)n^{\mathcal{O}(1)}$ et sont dans la classe EPTAS (pour Efficient Polynomial-Time Approximation Scheme). Il a été montré que si un problème est dans la classe EPTAS, alors il est aussi dans la classe FPT (l'inverse est par contre faux).

Nous avons donc, en supposant P $\neq$ NP, l'implication suivante :

$$\text{FPTAS} \subseteq \text{EPTAS} \subseteq \text{PTAS} \subseteq \text{APX}$$

Comme relevé par Fellows [Fel02], prouver qu'un algorithme appartient à la classe PTAS est loin d'être efficace en pratique, même si l'idée d'avoir un algorithme polynomial permettant d'être aussi proche que voulu de la solution semble séduisant. En effet, il relève dans la littérature un PTAS pour une variante du problème MINIMUM TRAVELLING SALESMAN PROBLEM en $\mathcal{O}(n^{\frac{3000}{\epsilon}})$ donnant pour 20% d'erreur un algorithme polynomial en $\mathcal{O}(n^{15000})$. Il relève également un PTAS pour un problème d'ordonnancement de tâches de complexité $\mathcal{O}(n^{(3mm!)^{\frac{m}{\epsilon}+1}})$ qui, pour $m = 4$ machines et 20% d'erreur donne un algorithme polynomial dont la complexité est d'environ $\mathcal{O}(n^{10000000000000000000000000000000000000000000000000000000000000000000})$. On relève que de tels remarques peuvent également être effectuées pour les problèmes de la classe FPT.

Comme dans les sections précédentes, on peut aussi prouver qu'un type d'approximation est improbable pour un problème donné. Comme des classes de complexités ont été définies, des réductions sont aussi possibles. Par exemple, un problème APX-Complet est APX-Difficile et appartient à la classe APX mais ne peut pas admettre de schéma d'approximation – il ne peut pas appartenir à la classe PTAS. C'est un des problèmes les plus difficiles de la classe APX.

Cependant, la définition de réduction polynomiale donnée dans la Section 1.2.2 n'est pas suffisante. En effet, celle-ci ne conserve pas forcement le ratio d'approximation. C'est pourquoi la *L-réduction* (pour Linear Reduction) a été introduite afin de contrôler l'éloignement de ce ratio pendant la réduction [PY91].

Étant donnée une instance I d'un problème d'optimisation P, on considère $OPT_P(I)$ comme étant la solution optimale et $A_P(I)$ comme étant la solution donnée par l'algorithme A_P d'approximation pour P. On parlera de L-réduction d'un problème P_1 vers un problème P_2 s'il existe une paire de fonctions polynomiales (f, g) et deux constantes strictement positives α et β telles que :

1. pour une instance I_1 de P_1, $f(I_1)$ est une instance de P_2 telle que $OPT_{P_2}(f(I_1)) \leq \alpha \times OPT_{P_1}(I_1)$,
2. $|OPT_{P_1}(I_1) - g(A_{P_2}(f(I_1)))| \leq \beta \times |OPT_{P_2}(f(I_1)) - A_{P_2}(f(I_1))|$.

Intuitivement, la fonction f transforme une instance de P_1 en une instance de P_2. La fonction g donne une solution à l'instance de P_1 depuis une solution pour l'instance créée par f. Les constantes α et β servent à contrôler l'éloignement des solutions afin de préserver les caractéristiques d'approximation de P_1.

Ces réductions permettent de prouver qu'un ratio d'approximation est impossible. Parallèlement aux résultats positifs d'approximation, on obtient en quelque sorte deux frontières. Essayer de les rapprocher est un aspect intéressant de la recherche sur les algorithmes d'approximations (voir Figure 1.16).

Plus d'informations sur les algorithmes d'approximation peuvent être trouvées dans la traduction française du livre de Vazirani [Vaz07] ou dans [ACG+99]. Il existe également un site internet [1] répertoriant pour des dizaines de problèmes d'optimisation les résultats connus sur leur approximation, positifs et négatifs.

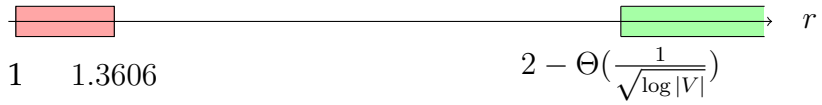


FIGURE 1.16 – Exemple de l'approximation du problème VERTEX COVER. Il est connu qu'on ne peut pas trouver de ratio d'approximation meilleur que 1.3606 [DS04] et on connaît un ratio d'approximation de $2 - \Theta(\frac{1}{\sqrt{\log |V|}})$ [Kar09]. Ce ratio n'est pas constant – on ne connaît pas de ratio constant meilleur que 2. Il semble important de tenter de rapprocher les deux limites afin de mieux connaître le comportement du problème.

1.3.5 Heuristiques

Tout comme un algorithme d'approximation, une *heuristique* s'exécute en temps polynomial et ne renvoie pas une solution exacte. Néanmoins, contrairement aux algorithmes sus-cités, on ne peut pas borner la marge d'erreur de l'algorithme – on peut être arbitrairement éloigné de la solution optimale.

Par exemple, pour répondre au problème MINIMUM TRAVELLING SALESMAN PROBLEM, on peut envisager un algorithme allant sur la ville la plus proche à chaque étape. Cet algorithme s'exécute en temps polynomial mais n'est pas optimal.

On propose en général des expérimentations montrant l'efficacité en pratique de l'algorithme – sur des données réelles, par rapport à d'autres heuristiques, ou par rapport à un algorithme exact pour des instances de petites tailles.

1. http://www.csc.kth.se/~viggo/problemlist/

1.3.6 Programmation linéaire

La *programmation linéaire* permet de résoudre de manière exacte des problèmes d'optimisation. Elle se décrit au moyen de variables, de contraintes à respecter sur ces variables et d'une fonction objectif. Le but est donc de donner des valeurs aux variables de manière à optimiser la fonction objectif, tout en respectant les contraintes. Plus d'informations sur cette technique peuvent se trouver dans le livre dont un des auteurs est l'inventeur d'un des premiers algorithmes de résolution efficace, le *simplex* [DT97].

Le programme suivant en montre un exemple :

$$\max x_1 + 2x_2 - x3 \tag{1.1}$$

$$\begin{cases} 2x_1 - 2x_2 + x3 \leq 2 \\ x_1 + x_2 + x_3 \leq 3 \\ x_1, x_2, x_3 \geq 0 \end{cases} \tag{1.2}$$

On trouve la fonction objectif en (1.1), tandis que les contraintes sont en (1.2).

Il est difficile de traduire les contraintes d'un problème NP-Complet avec de la simple programmation linéaire – en effet, cette dernière peut être résolue en temps polynomial. Il est par contre plus facile d'exprimer un problème sous forme de *programmation entière*, qui est équivalent à la programmation linéaire excepté le fait que les variables doivent être des nombres entiers au lieu de rationnels. Résoudre une programmation entière est par contre un problème NP-Complet.

Une technique courante dans la recherche d'algorithmes d'approximation consiste à décrire un problème NP-Complet sous forme de programmation entière, puis de *relâcher* les contraintes entières pour obtenir un programme de programmation linéaire ne donnant donc pas de solution exacte au problème. Le point délicat est de pouvoir borner l'erreur faite lors de la relaxation des variables [Vaz07].

Enfin, une version pouvant être dite intermédiaire est la *programmation pseudo-booléenne* (ou 0-1 programmation entière). Elle est équivalente à la programmation entière mais les variables peuvent cette fois uniquement prendre des valeurs booléennes, 0 ou 1. On peut assez simplement exprimer le problème VERTEX COVER pour un graphe $G = (V, E)$ à l'aide de programmation pseudo-booléenne :

$$\min \sum_{v \in V} x_v \tag{1.3}$$

$$\begin{cases} \forall \{u, v\} \in E, & x_u + x_v \geq 1 \\ \forall v \in V, & x_v = \{0, 1\} \end{cases} \tag{1.4}$$

Ici, il existe une variable booléenne pour chaque sommet du graphe. Celle-ci vaut 1 si le sommet est dans la solution, 0 sinon. La fonction en (1.3) consiste donc à minimiser le nombre de sommets dans la solution, avec la condition que chaque arête $\{u, v\}$ du graphe soit couverte par au moins un des deux sommets (1.4).

De nombreux *solveurs* efficaces existent pour résoudre ces types de problèmes, même avec des milliers de variables et de contraintes. Ils s'affrontent même régulièrement lors de compétitions internationales. Ces programmes sont en effet d'importance car la programmation linéaire trouve des applications dans de très nombreux domaines (flux de transports, planification, production pétrolière, télécommunication...). L'auteur du premier algorithme de résolution de programmation linéaire évoque l'exemple suivant. Le nombre de configurations possibles pour l'affectation à 70 hommes de 70 tâches dépasse le nombre de particules de l'univers. Mais l'exprimer en programmation linéaire ne demande alors que quelques instants pour le résoudre [DT97].

S'il semble que les premiers systèmes de programmation linéaire remontent à Fourier au XIXème siècle, la programmation linéaire a été véritablement utilisée pendant la seconde guerre mondiale, afin d'optimiser les envois de matériels et réduire les coûts. Elle était cependant alors gardée secrète et c'est après 1947 que son utilisation a véritablement explosée, notamment grâce à ces applications dans l'industrie [DT97].

Des solveurs spécifiques à la programmation entière ou pseudo-booléenne existent et sont en général les plus efficaces car plus optimisés. Malheureusement, il est souvent difficile de déterminer un solveur plus efficace de manière générale car ceux-ci se comportent différemment selon l'instance à traiter.

1.3.7 Algorithmes exponentiels exacts

La complexité d'un problème NP-Complet est forcement exponentielle. Si la complexité paramétrée cherche à concentrer la partie exponentielle selon un paramètre petit par rapport à la taille de l'instance, un domaine actuel de recherche consiste à diminuer l'exposant de la partie exponentielle de la complexité. Ceci selon le principe évident qu'un problème de complexité $\mathcal{O}(1.5^n)$ sera beaucoup plus performant qu'un problème de complexité $\mathcal{O}(2^n)$ comme le montre la Table 1.3. On remarque d'ailleurs qu'un algorithme exponentiel en $\mathcal{O}(1.1^n)$ est plus performant qu'un algorithme polynomial en

$\mathcal{O}(n^{10})$, tout comme un algorithme en $\mathcal{O}(1.5^n)$ pour des valeurs de n inférieures à 100. Par exemple, pour le problème INDEPENDENT SET, l'algorithme naïf consistant à tester tous les sous-ensembles de sommets pouvant potentiellement être dans la solution mène à une complexité bornée par 2^n (chacun des n sommets est ou non – deux choix – dans la solution). Il a été montré qu'on pouvait résoudre ce problème en temps $\mathcal{O}(1.2109^n)$.

$n \backslash f(n)$	n^5	n^{10}	1.05^n	1.1^n	1.5^n	2^n	5^n	$n!$
10	$<$ 1 sec	9 sec	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec
20	$<$ 1 sec	2 h	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	1 jour	71 ans
50	$<$ 1 sec	> 2 ans	$<$ 1 sec	$<$ 1 sec	0.6 sec	12 jours	$> 10^{18}$ ans	$> 10^{47}$ ans
100	9 sec	3000 ans	$<$ 1 sec	$<$ 1 sec	12 ans	$> 10^{13}$ ans		
200	5 min	3 millions d'années	$<$ 1 sec	0.2 sec	$> 10^{18}$ ans			
500	8 h		36 sec	> 14000 ans				

TABLE 1.3 – Temps estimé pour accomplir n opérations pour un algorithme de complexité $f(n)$, considérant un processeur effectuant 2^{30} opérations par secondes. Tableau adapté depuis celui de [Gas10].

Le lecteur intéressé par ce sujet pourra se reporter sur l'article de revue de Woeginger [Woe01] ou le récent livre de Fomin et Kratsch [FK10].

Chapitre 2

Quelques notions de biologie

Contenu

Les problèmes étudiés durant cette thèse sont motivés par des problématiques biologiques. Malheureusement, comme dans la plupart des domaines, ces dernières se modélisent bien souvent par des problèmes difficiles du point de vue algorithmique. Bien que cette thèse s'intéresse à l'aspect algorithmique de ces problématiques, nous présentons brièvement dans ce chapitre les quelques notions de biologie nécessaires à leur compréhension.

La *génétique* représente la science étudiant la transmission des caractères héréditaires entre générations d'êtres vivants. Ce chapitre introduit les acteurs de la génétique en suivant le schéma du dogme central de la biologie moléculaire, avant de présenter les réseaux biologiques, qui mettent en relation ces acteurs.

2.1 Le dogme central de la biologie moléculaire

Défini en 1958 par F. Crick (1916 – 2004), le dogme central de la biologie moléculaire représente schématiquement le modèle de la conservation et l'utilisation de l'information génétique. Il se résume dans sa version initiale de la manière suivante : l'ADN dirige sa propre réplication (à l'identique, aux mutations près), ainsi que sa transcription en ARN, ce dernier pouvant être traduit en protéines (voir Figure 2.2).

Ce dogme a été précisé depuis. En effet, la théorie initiale n'envisageait pas la possibilité que l'ARN puisse être capable de se répliquer ni de rétrotranscrire de l'ADN – ce qui a été mis en évidence ensuite chez certains virus [Cla05].

Si le terme *gène* est proposé en 1909 par le Danois W. Johannsen (1857 – 1927), les travaux tentant de mettre en évidence l'hérédité des caractères sont dus au Français J. B. de Lamarck (1744 – 1829) avec la théorie du transformisme, puis à C. Darwin (1809 – 1882) et H. M. de Vries (1848 – 1925) avec la théorie de l'évolution, mettant en évidence une transformation progressive dues à des mutations et des sélections naturelles.
En 1866, le Tchèque G. Mendel (1822 – 1884) observe sept caractères du pois (tels que la forme des graines ou la couleur des cotylédons) pouvant prendre deux valeurs possibles, sur plusieurs générations (voir Figure 2.1). Il énonce les lois dites de Mendel, considérées comme la base de la génétique moderne, mettant en évidence la transmission des caractères héréditaires. Malheureusement, le bien-fondé de ses travaux ne fut reconnu qu'après sa mort.

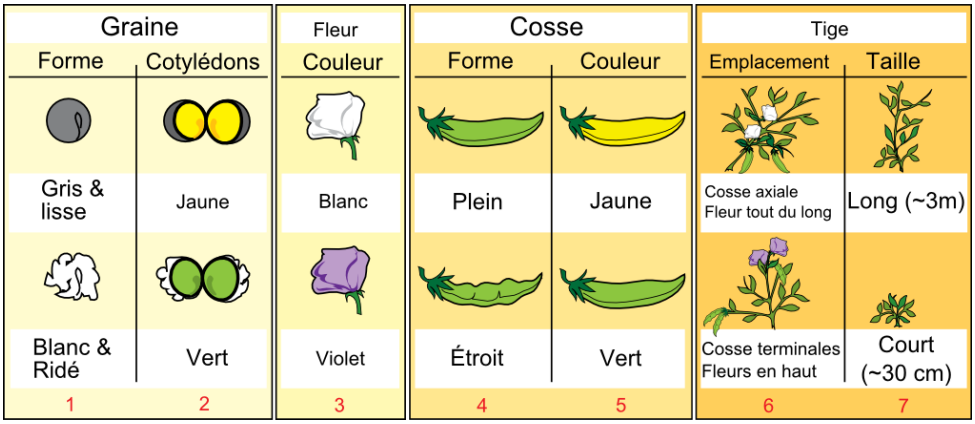


FIGURE 2.1 – Caractéristiques étudiés par Mendel sur des pois. Source Wikipedia.

Nous présentons maintenant les différentes entités intervenant dans ce dogme ainsi que les différents processus évoqués – le lecteur cherchant plus d'informations pourra consulter l'un des nombreux livres sur la biologie moléculaire, la biochimie ou la génétique, comme [Cla05] ou [AJL+07].

2.1.1 L'ADN

L'*Acide DésoxyriboNucléique* (ADN) fut découvert en plusieurs étapes par différents chercheurs, entre 1869 et 1928. C'est une *molécule* (assemblage chimique de plusieurs *atomes*), que l'on retrouve dans toutes les *cellules* vivantes et chez certains virus. Il contient l'ensemble de l'information génétique nécessaire au fonctionnement d'un organisme. Il est également responsable de la transmission de cette information. La molécule d'ADN est en forme de double hélice (voir Figure 2.3), où chaque hélice (c'est-à-dire chaque *brin*) est constituée d'une suite de millions de *nucléotides*. Il existe quatre types de nucléotides, différenciés par une *base azotée* variable nommée Adénine

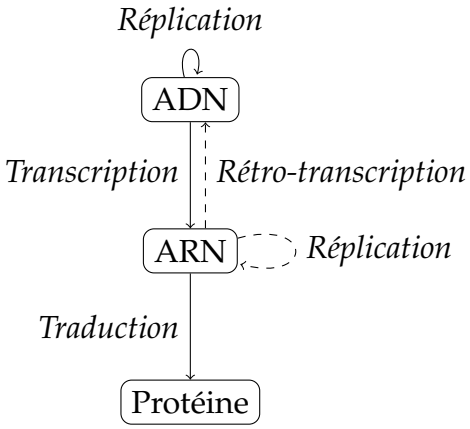


FIGURE 2.2 – Schéma du dogme central de la biologie moléculaire, où les pointillés représentent les transferts peu courants d'informations.

Le terme *dogme* est utilisé par F. Crick mais peut prêter à confusion (il vient d'un malentendu car F. Crick ne connaissait pas sa signification exacte). Il s'agit d'une hypothèse scientifique plutôt qu'une affirmation incontestable.

(A), Cytosine (C), Guanine (G) et Thymine (T). Elles sont complémentaires ; ainsi, dans la structure en double hélice, un A sera toujours en face d'un T, et un G en face d'un C. Elles sont liées par des *ponts hydrogènes* (*appariement*), au nombre de deux pour la paire A-T, de trois pour la paire G-C.

L'ADN est répliqué lors de la *méiose*, donnant deux molécules identiques (aux mutations près). L'information se transmet ainsi d'une cellule mère à deux cellules filles. Lors de cette réplication, la double hélice s'ouvre telle une fermeture à glissière libérant les deux brins complémentaires par l'action de protéines, comme illustré Figure 2.4.

Deux nouvelles molécules d'ADN sont alors construites, chacune composée de deux brins : un de l'ancienne molécule et un nouvellement construit à partir de nucléotides complémentaires du premier. Ce nouveau brin est construit par l'action de protéines synthétisant les nucléotides complémentaires à celles du brin de l'ancienne molécule.

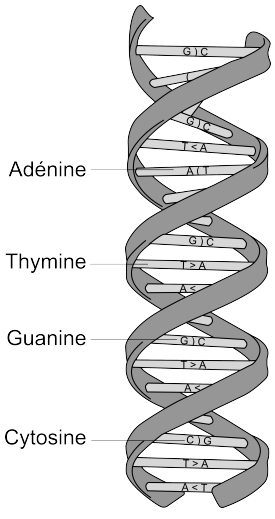


FIGURE 2.3 – Structure en double hélice de l'ADN. Source Wikipedia.

Le séquençage de l'ADN humain (c'est-à-dire la détermination de la succession des nucléotides le composant) a été achevé dans les années 2000. Il a été le fruit d'une quinzaine d'années de coopérations scientifiques internationales (*Human Genome Project*),

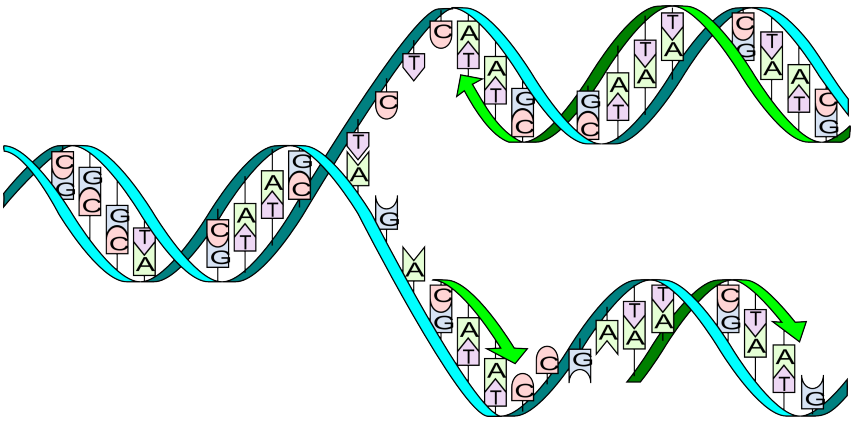


FIGURE 2.4 – Schéma de la réplication de l'ADN. La double hélice s'ouvre pour permettre la formation de deux nouvelles molécules. Source Wikipedia.

déterminant les trois milliards de paires de nucléotides de l'ADN humain.

Un récent article indique qu'un cinquième des séquençages effectués serait contaminé par de l'ADN humain, suite au manque de rigueur des laboratoires exécutant le séquençage [LOO11]. De l'ADN humain a été trouvé dans les résultats du séquençage des génomes de plantes, de virus et d'animaux obtenus depuis des bases de données publiques.

2.1.2 L'ARN

L'*Acide RiboNucléique* (ARN) est une molécule proche chimiquement de l'ADN. C'est une suite de nucléotides qui résulte de la transcription d'une partie de l'ADN. La transcription est le processus par lequel une séquence d'ARN est créée, complémentaire à une partie d'un brin d'ADN, où la Thymine (T) est remplacée par l'Uracile (U). L'ARN est plus court que l'ADN (au plus quelques milliers de nucléotides) et le plus souvent à simple brin. Sa durée de vie est généralement courte.

On trouve différents types d'ARN, codant différents types d'informations :

– celles nécessaires au codage de la fonction des protéines,

– celles nécessaires au codage de sa propre fonction.

Si seule la suite de nucléotides est utilisée pour représenter l'ARN – on parle de *structure primaire*.

Cependant, même si elle est généralement formée d'un simple brin, la molécule d'ARN tend à se replier sur elle-même pour former des *liaisons hydrogène* entre des couples de ses propres nucléotides (voir Figure 2.5, en a)). Celles-ci sont souvent entre bases complémentaires (comme pour l'ADN), mais il arrive d'observer des liaisons

entre bases non-complémentaires, qui semblent jouer un rôle très important [VM00]. L'ensemble de ces liaisons représente la *structure* de l'ARN, que l'on appelle *structure secondaire*. Cette dernière fournit l'information sur les liaisons hydrogène, en complément de la séquence de nucléotides. Il est largement admis que considérer la structure secondaire est important pour étudier les ARN ne codant pas de protéines [AJL+07].

La *structure tertiaire* fait référence au repliement de l'ARN dans l'espace à trois dimensions (voir Figure 2.5, en c)).

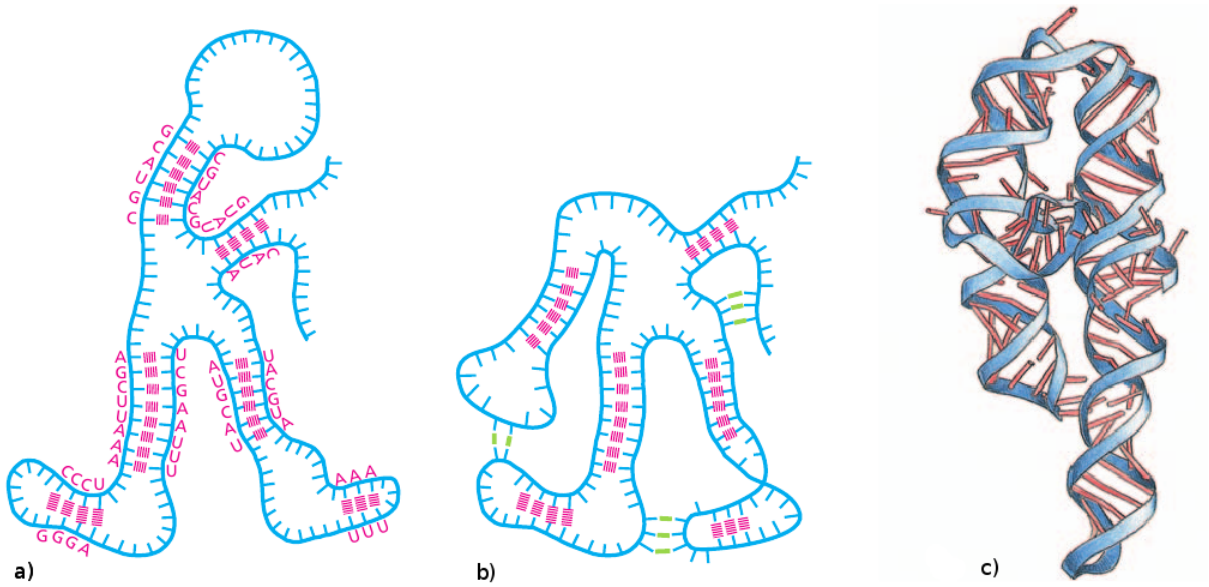


FIGURE 2.5 – Illustration du repliement de l'ARN. En a), la molécule d'ARN s'est repliée sur elle-même avec des liaisons hydrogène. En b), l'ARN se replie avec les liaisons hydrogène comme en a) en rouge et des pseudo-nœuds en vert. En c), une vue en trois dimensions du repliement d'une molécule d'ARN. Source [AJL+07].

Étant donnés quatre nucléotides aux positions i, j, k, l sur la séquence d'ARN, où $i < j < k < l$, il y a un *pseudo-nœud* s'il y a une liaison hydrogène entre les nucléotides aux positions i et k, et une liaison hydrogène entre les positions j et l. Les liaisons vertes sur la Figure 2.5 en b), sont des pseudo-nœuds. Intuitivement, si la séquence d'ARN était disposée sur une droite, les liaisons n'étant pas des pseudo-nœuds représenteraient des arcs de cercles imbriqués ou côte à côte, tandis que les pseudo-nœuds impliqueraient des croisements d'arcs (voir Figure 2.6).

La structure tertiaire est considérée comme étant la structure la plus proche de la réalité, puisqu'elle décrit l'organisation spatiale de l'ARN et contient des liaisons supplémentaires. De plus, la présence de pseudo-nœuds est fréquente et joue un rôle important dans la fonction de l'ARN [SB05]. Néanmoins, il est beaucoup plus difficile d'obtenir une telle structure.

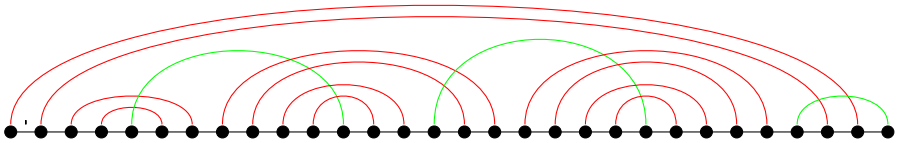


FIGURE 2.6 – Représentation de la structure des liaisons hydrogène de l'ARN représenté Figure 2.5, en b). La séquence de nucléotides est disposée sur une droite. Pour des raisons de lisibilité, chaque bloc de liaisons hydrogène dessinées en rouge est représenté par deux arcs rouges, tandis que chaque bloc de liaisons hydrogène dessinées en vert est représenté par un seul arc vert.

2.1.3 Les protéines

Les *protéines* sont, comme l'ADN et l'ARN, des molécules, considérées comme les plus complexes et variées d'un être vivant (il est estimé qu'un être humain fabrique au moins 100 000 types de protéines différentes [Pev06]). Une protéine est constituée d'une ou plusieurs chaînes d'acides aminés (petites molécules) – on en connaît (actuellement) 22 qui composent les protéines.

Tout comme l'ARN, la protéine possède (au moins) trois structures (voir Figure 2.7) :

- la structure primaire, où seule la séquence d'acides aminés est considérée,
- la structure secondaire, où en plus de la séquence, on considère le repliement local des acides aminés (avec des liaisons hydrogène), sous forme d'hélice (*hélice alpha*), ou sous forme de feuillet (*feuillet beta*),
- la structure tertiaire, où on considère l'agencement dans l'espace entre ces hélices, ces feuillets, et des segments sans structure particulière.

Les protéines résultent de la traduction d'un ARN particulier, l'ARN messager (ARNm), en acides aminés. On appelle *codon* un groupe de trois nucléotides représentant, selon le code génétique universel (valable pour la quasi-totalité des êtres vivants, voir Figure 2.9), soit un acide aminé unique, soit une instruction.

Schématiquement, un *ribosome* (petit organite constitué de protéines et d'ARN) se fixe sur l'ARN messager. Une fois qu'il est parvenu au codon initiateur AUG, la traduction commence – le ribosome parcourt alors le brin d'ARNm codon par codon. Chaque codon correspond à un acide aminé, qui est apporté par l'ARN de transfert (ARNt) portant le codon complémentaire et qui se fixe à l'ARNm par l'intermédiaire du ribosome. Ce processus continue jusqu'à la rencontre d'un codon STOP, arrêtant la traduction pour la protéine courante.

Les protéines ont diverses fonctions, par exemple la *catalyse* (les *enzymes* accélèrent des réactions chimiques), le *transport* (de l'oxygène pour l'*hémoglobine*, du fer pour la *transferrine*...), la *communication entre cellules* (entre autres, l'*insuline*, secrétée par le pancréas lorsqu'il y a trop de sucre dans le sang, permet le stockage du sucre en surplus dans les cellules) ou la *réponse immunitaire* (notamment les *anticorps* luttant contre les corps étrangers). On note qu'une même protéine peut assumer plusieurs

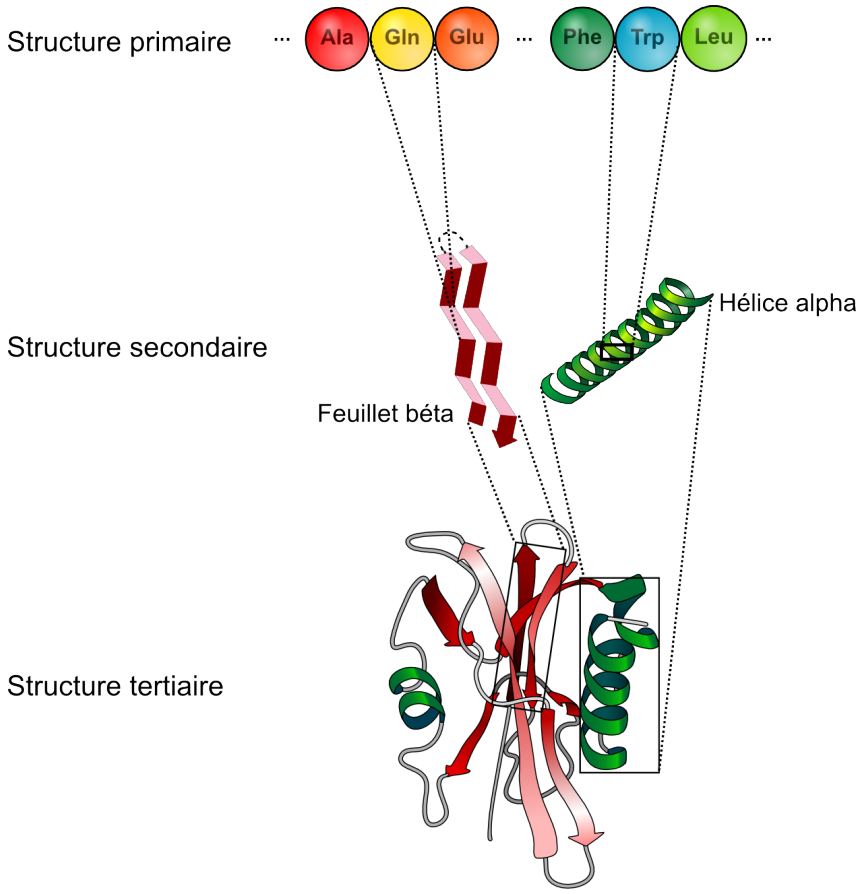


FIGURE 2.7 – Représentation de la structure primaire, secondaire et tertiaire d'une protéine.

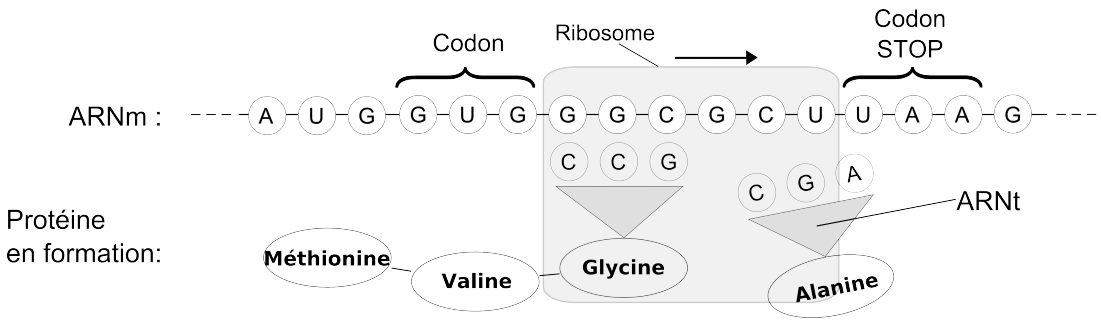


FIGURE 2.8 – Schéma de la traduction de l'ARNm en protéine.

fonctions [Jef99]. Par exemple, la lactoferrine, qui est présente dans le lait de vache comme dans celui de la femme, mais aussi dans les larmes, la bile ou la salive, orchestre diverses fonctions telles que l'absorption du fer, une activité immunitaire, ou encore la régulation de la croissance osseuse [PLM09].

Deux protéines *homologues* ont des origines communes car elles proviennent d'un ancêtre commun. Par conséquent, leur fonction est semblable. L'homologie peut être reconnue selon une séquence d'acides aminés similaire et une structure spatiale proche. En pratique, on détermine l'homologie de deux protéines *via* des programmes informatiques spécifiques, donnant un score de similarité. Les protéines sont considérées comme homologues si ce score dépasse un certain seuil.

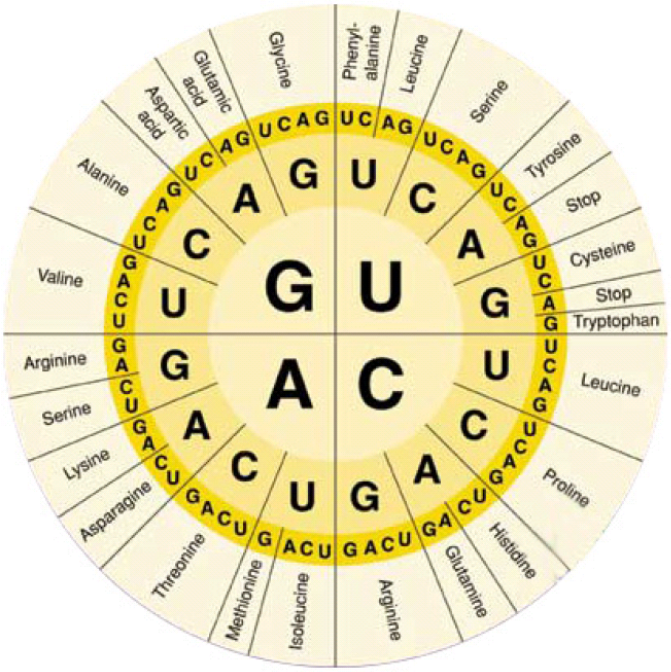


FIGURE 2.9 – Code génétique universel, traduisant sous forme d'acides aminés trois nucléotides lues. On remarque qu'un même acide aminé est codé par plusieurs codons différents. Source Wikipedia.

2.1.4 Les gènes

Le *gène* est l'unité de transcription. L'ensemble de l'information génétique d'un individu est appelé *génome*. Ce dernier contient l'ensemble des gènes, mais aussi des portions d'ADN non codantes (c'est-à-dire n'étant pas traduites en protéines).

Le gène est une portion d'ADN, un enchaînement de nucléotides portant l'information génétique à l'origine de la synthèse d'une protéine. Chez les *eucaryotes* (organismes avec un noyau dans leurs cellules), c'est une succession d'*exons* (séquence codante) et d'*introns* (séquence non codante). Chez ces eucaryotes, l'ARNm subit une maturation qui comprend un processus d'*épissage* : les introns ainsi que certains exons sont retirés de l'ARNm, permettant une multitude de combinaisons. Ainsi, un gène, suivant la combinaison exécutée, peut produire plusieurs protéines (c'est pourquoi il existe plus de protéines que de gènes chez l'homme).

Chaque cellule contient l'ensemble des gènes. Néanmoins, ils ne sont pas tous exprimés (c'est-à-dire traduits en protéines) dans chaque cellule. On dit alors qu'un gène est *activé* ou non.

On estime à environ 12.000 le nombre de gènes chez la mouche drosophile, 25.000 chez l'homme et 45.000 pour le riz [Cla05]. Ceci prête à supposer que notre relative complexité ne vient pas directement de notre nombre de gènes. Il y a quelques années, certains pensaient que l'homme portait environ 100.000 gènes [Cla05] ! Le nombre de 25.000 devrait maintenant rester stable car tous ses gènes semblent avoir été identifiés. Cependant, déterminer la fonction de chacun d'entre eux reste une tâche importante.

L'humain a depuis longtemps, la réputation d'avoir un odorat moins développé que les autres mammifères. La détermination des génomes explique pourquoi. En effet, les mammifères ont un ensemble d'une centaine de gènes proches, relatifs aux récepteurs olfactifs. Si chez la souris la quasi-totalité de ces gènes sont actifs, ils ne sont que 30% chez l'humain, soit le plus bas taux chez les mammifères [Cla05].

2.2 Les réseaux biologiques

2.2.1 Généralités

Les molécules présentées à la section précédente peuvent être considérées comme le premier barreau d'une échelle complexe. En effet, ces molécules font fonctionner une cellule, qui fait elle-même partie d'un organe. Un ensemble d'organes forme un organisme et un ensemble d'organismes forme un écosystème. Tous ces éléments interagissent et évoluent au cours du temps. Énumérer les éléments intervenant dans les mécanismes cellulaires ne suffit pas pour comprendre ce dernier, il faut également étudier les interactions entre ces éléments. Kitano [Kit02] fait l'analogie suivante : identifier les gènes ou les protéines d'un organisme est comme énumérer les composants d'un avion. Cela ne suffit pas à comprendre la complexité sous-jacente de l'objet en question, il faut comprendre comment ces différents éléments sont assemblés. Le lecteur désirant plus d'informations sur les réseaux biologiques peut se tourner vers [AA03] ou [JS08].

Pour représenter ces interactions de manière abstraite, on modélise un *réseau biologique* par un graphe (Section 1.1). Les éléments observés représentent les sommets du graphe tandis que les interactions entre les éléments correspondent à des arêtes (ou arcs) entre les sommets.

Les interactions existant à différents niveaux, elles sont représentées par des réseaux biologiques à chacun d'entre eux. Au sein d'une protéine, le réseau peut représenter les interactions atomiques menant au repliement de la protéine sur elle-même. À un niveau intermédiaire, les molécules interagissent entre elles. Enfin, un réseau peut représenter des organismes interagissant entre eux au sein d'une population.

Parmi les différents réseaux d'interactions entre les molécules, nous étudions plus particulièrement :

- les réseaux métaboliques,
- les réseaux de régulation de gènes,
- les réseaux d'interactions entre les protéines.

Un exemple de réseau biologique d'interactions moléculaire est donné Figure 2.10.

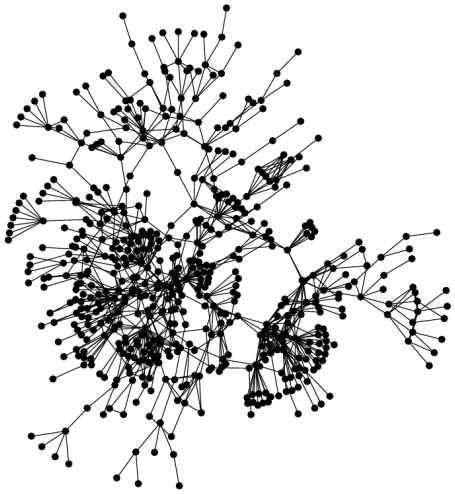


FIGURE 2.10 – Exemple de réseau biologique : une composante connexe d'un réseau d'interactions entre protéines de l'homme. Source : [JS08].

2.2.2 Trois réseaux biologiques d'interactions moléculaires

Nous détaillons ici les trois principaux types de réseaux d'interactions biologiques, au niveau moléculaire.

Les *réseaux métaboliques* mettent en jeu les métabolites (composés) et leurs réactions biochimiques dans un organisme. Les métabolites sont usuellement de petites molécules comme du glucose ou des acides aminés. Une réaction biochimique prend des métabolites en entrée, appelés *substrats*, pour produire des métabolites en sortie, appelés *produits*. Si la réaction n'est pas spontanée, elle nécessite un catalyseur. Ce dernier est une enzyme qui a pour rôle d'accélérer la réaction. Le réseau (exemple Figure 2.11) peut être représenté de plusieurs manières équivalentes :

- Par un *graphe des composés*, où les sommets sont les métabolites et où il y a un arc (orienté) entre le substrat et le produit, annoté par l'enzyme catalysant leur réaction.
- Par un *graphe biparti*, où deux types de sommets sont définis, un pour les métabolites et un pour les enzymes. Une réaction est donc constituée de deux arcs, allant du substrat au produit en passant par l'enzyme. Le graphe ainsi défini est biparti.
- Par un *graphe des réactions*, où les sommets sont les réactions et où il y a un arc entre deux réactions (enzymes) s'il existe un métabolite qui est produit par l'une et consommé par l'autre.

Les *réseaux de régulation de gènes* modélisent l'activation des gènes dans la cellule. Ces réseaux aident à comprendre pourquoi l'ensemble des gènes ne s'exprime pas dans la cellule à un instant donné. Certains gènes régulent la transcription – processus produisant l'ARN depuis l'ADN – par des actions d'inhibition et d'activation, ce sont des *gènes régulateurs* [Cla05]. En effet, ces gènes peuvent contenir l'information pour la traduction d'une protéine spécifique, une protéine activatrice. Cette dernière peut stimuler la transcription d'un ou plusieurs gènes. À l'inverse, ces gènes peuvent contenir

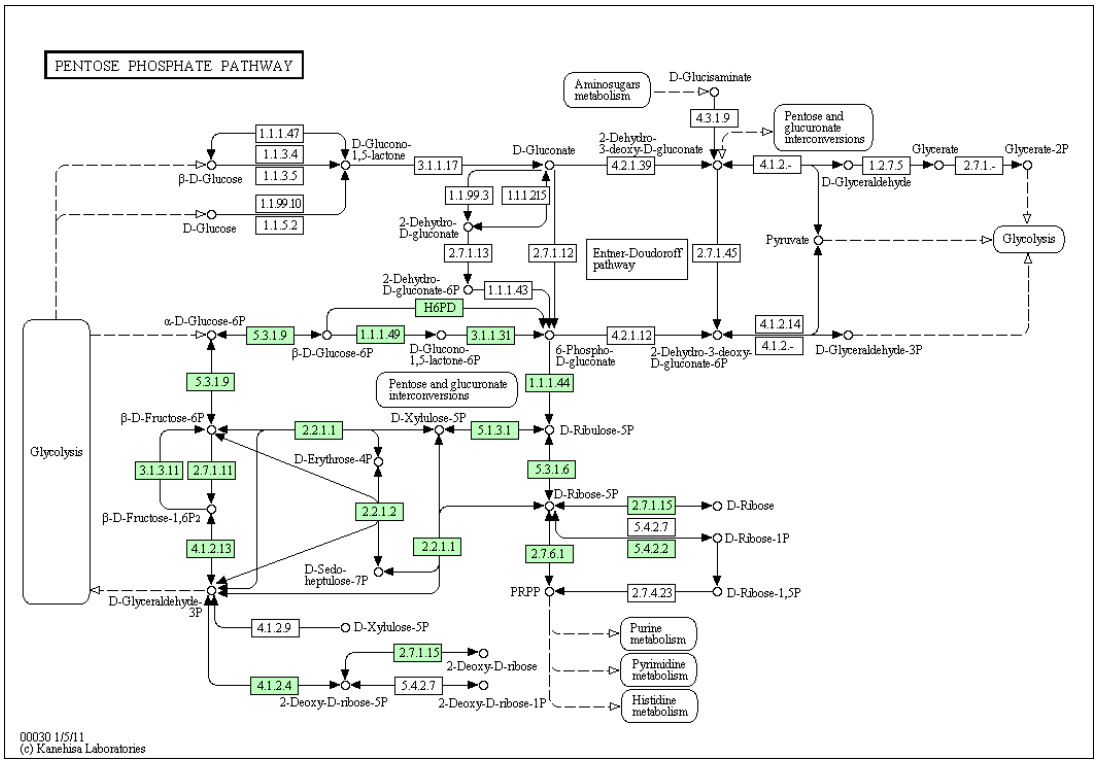


FIGURE 2.11 – Extrait d'un réseau métabolique : la voie des pentoses phosphates chez l'homme. Les rectangles représentent les enzymes catalysant la réaction, le code indiqué étant leur nomenclature EC (Enzyme Commission). Figure obtenue depuis la base de donnée KEGG.

l'information pour la traduction d'une protéine répressive, où cette dernière peut empêcher la transcription d'un gène. Dans les cellules d'organismes complexes, on observe des séries de gènes régulateurs, où un gène régulateur régule un autre gène régulateur. Ainsi, ce réseau est modélisé par un graphe orienté, où les sommets sont les gènes et les arcs représentent les régulations effectuées par les gènes. On note qu'il est difficile d'obtenir des réseaux de régulation de gènes complets. Ce type de réseau ne prend en compte que la régulation de la transcription, alors que la régulation d'un gène peut en fait intervenir à d'autres moments que la transcription. Il serait donc plus juste de les nommer réseaux de régulation de la transcription – ce n'est malheureusement généralement pas le cas dans la littérature.

Enfin, les *réseaux d'interactions entre protéines* modélisent les interactions physiques entre les protéines. Dans un tel réseau (exemple Figure 2.10), les sommets représentent les protéines tandis qu'une arête entre deux sommets représente une interaction physique. En effet, certaines protéines proches dans l'espace ont aussi une complémentarité physique. Ces interactions peuvent être :

- *ioniques*, reliant deux atomes de charges électriques opposées,
- *hydrogènes*, reliant deux atomes électronégatifs (c'est-à-dire de charges négatives) par un atome d'hydrogène,
- *hydrophobes*, des poches hydrophobes à l'abri de l'eau ayant tendance à se regrouper,

– dues aux *forces de van der Waals*, interactions électrostatiques sans intérêt biologique car faibles et non spécifiques.

Beaucoup de fonctions impliquent plus d'une protéine – on observe en effet des associations entre plusieurs protéines, formant des *complexes protéiques*. Ces interactions peuvent être observées au sein même de la cellule. Elles peuvent être dues à une information venant de l'extérieur de la cellule. Par exemple, l'insuline, protéine véhiculée par le sang, se fixe sur une protéine réceptrice membranaire de la cellule cible (la membrane sépare la cellule de son environnement). Cette interaction va provoquer l'activation de protéines à l'intérieur de la cellule telles que la protéine IRS-1 (pour Insulin Receptor Substrate). Ces interactions, dues à une concentration de glucose trop élevée dans le sang, vont permettre de diminuer celle-ci.

2.2.3 Obtention, stockage et qualité des données

Selon Sharan et Ideker [SI06], les données disponibles concernant les interactions utilisées dans les réseaux augmentent de manière exponentielle. Selon ces mêmes auteurs, en 2001, seules quelques centaines d'interactions étaient connues pour un organisme donné, alors qu'en 2006, des milliers de données étaient disponibles pour les organismes très étudiés. Ceci s'explique en partie par le fait que jusqu'au début des années 2000, les informations étaient récupérées presque une à une. Depuis, des méthodes globales existent et permettent l'acquisition de plusieurs interactions simultanément. Une analyse comparative sur la qualité de ces nouvelles méthodes pour les interactions entre les protéines se trouve dans [vMKS+02].

L'augmentation rapide du nombre de ces données impose des méthodes efficaces pour les stocker et les rendre disponibles à la communauté. Ainsi, des dizaines de bases de données existent (le premier chapitre de la thèse de Bader recense et décrit une soixantaine d'entre elles [Bad03]), parfois généralistes, souvent spécifiques à un type d'interaction ou à un organisme.

Finley [1] maintient une liste des plus populaires. Citons par exemple BIND (Biomolecular Interaction Network Database – `http://www.bind.ca/`), DIP (Database of Interacting Proteins – `http://dip.doe-mbi.ucla.edu/`), KEGG (Kyoto Encyclopedia of Genes and Genomes – `http://www.genome.jp/kegg/`) ou MINT (Molecular INTeraction database – `http://mint.bio.uniroma2.it/`).

L'inconvénient de ces nouvelles méthodes traitant plus d'une interaction à la fois est la présence importante de bruit [vMKS+02]. Concernant les interactions entre les protéines, on estime que dans les bases de données actuelles, la moitié des informations présentes ne devrait pas y être (*faux positifs*) tandis qu'il manquerait la moitié des informations (*faux négatifs*) [EKJ+02].

Une des réponses algorithmiques à ce problème consiste à utiliser des graphes

1. `http://proteome.wayne.edu/PIDBL.html`

pondérés pour modéliser ces réseaux où le poids d'une arête symbolise le taux de confiance de l'interaction correspondante.

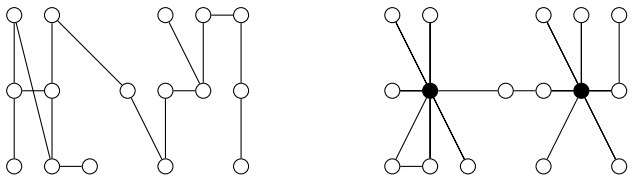
2.2.4 Topologie des réseaux

Une fois ces informations récupérées et mises en relation au sein d'un réseau, il semble naturel de s'interroger sur la topologie et les propriétés du graphe obtenu.

Les graphes dits *scale-free* ont beaucoup de sommets avec un degré faible et peu de sommets avec un très haut degré – la distribution des degrés suit une loi de puissance. Les sommets de degré élevé jouent le rôle de *hubs* (échangeurs) – la plupart des chemins liant deux sommets quelconques du graphe passent par ces hubs (voir Figure 2.12).

Il a été montré que le réseau Internet était scale-free. Cette topologie offre l'avantage d'une communication plus rapide entre les nœuds et d'être moins sensible aux pannes aléatoires qu'un réseau avec distribution aléatoire des arêtes. Ceci est basé sur le fait que sur un réseau avec distribution aléatoire des arêtes, la panne d'un nœud a de grandes chances de déconnecter le réseau. Tandis que sur un réseau scale-free, les nœuds déconnectant le réseau sont ceux à haut degré – ceux-ci sont peu nombreux donc la probabilité d'une panne critique pour le réseau est plus faible.

Certains réseaux sociaux sont également scale-free. Par exemple, dans le réseau des collaborations scientifiques, un hub typique est Paul Erdös car il a écrit des articles avec plus de 500 auteurs différents.
Dans celui des acteurs ayant joué ensemble dans des films à Hollywood, la carrière prolifique de l'acteur Kevin Bacon fait de lui un hub.



$a)$ $b)$

FIGURE 2.12 – En $a)$, un exemple de graphe où les arêtes sont définies de manière aléatoire. En $b)$, un exemple de graphe *scale-free*, où certains sommets, les hubs dessinés en noir, ont un degré supérieur aux autres.

Cette notion de graphe scale-free est liée à celle (contestée) des *graphes petits mondes*, où la distance entre toute paire de sommets est bornée (le *diamètre* du graphe est borné).

Au début des années 2000, certains articles ont appliqué ces notions aux réseaux biologiques – des expériences de comptages semblant abonder dans ce sens (par exemple

Initialement, la notion des graphes petits mondes est liée au *paradoxe de Milgram* (car contraire à l'intuition), supposant que n'importe quel individu est relié par une chaîne d'en moyenne six relations à n'importe quel autre individu. Deux personnes, choisies au hasard, se trouveraient à seulement six « poignées de mains » l'une de l'autre, six intermédiaires suffiraient pour qu'elles se rencontrent.

[JTA+00]). Les réseaux métaboliques et d'interactions entre les protéines seraient scale-free. Certains sommets auraient un degré plus important que lorsque la répartition des arêtes de ces réseaux est aléatoire. Ces hubs représenteraient par exemple des protéines ayant une fonction plus importante que d'autres [JMBO01]. Les réseaux métaboliques seraient également petit monde, le diamètre du réseau serait borné et varierait peu entre deux espèces [AA03].

Cependant, beaucoup de critiques ont été émises sur cette application systématique d'un modèle générique à des réseaux particuliers (comme souvent en biologie). Parmi celles-ci, on retrouve le fait que les réseaux métaboliques sont souvent incomplets – ainsi, si un extrait de ce réseau est scale-free, cela ne donne pas d'évidence directe quant au caractère scale-free du réseau complet. En outre, si tous les nœuds du réseau d'Internet sont identiques, ce n'est pas le cas pour les sommets d'un réseau biologique. Enfin, les réseaux biologiques sont si bruités qu'il semble délicat de donner des propriétés sur leur topologie. L'article de Alm et Arkin [AA03] ainsi que la Section 2.4.2 de la thèse de Vincent Lacroix donnent plus d'informations et de références sur cette problématique [Lac07].

2.2.5 Évolutions des réseaux

Au cours du temps, les organismes sont sujets à l'*évolution*, à une modification de leurs propriétés, menant parfois à la *spéciation*, processus par lequel de nouvelles espèces apparaissent. Cette évolution résulte essentiellement de la *dérive génétique*, c'est-à-dire des remaniements des génomes et des mutations. Les traits favorisant la survie d'une espèce à un environnement sont favorisés, c'est la *sélection naturelle*. Cette théorie est basée sur le livre de Charles Darwin [Dar59].

Un modèle largement utilisé pour représenter les relations de parenté entre les espèces étant susceptibles d'avoir un ancêtre commun est l'*arbre phylogénétique* (voir Figure 2.13). Dans un tel arbre, les feuilles représentent les espèces actuelles tandis qu'un nœud interne est considéré comme l'ancêtre de ses descendants. Cependant, il a été montré qu'un organisme pouvait intégrer du matériel génétique venant d'autres organismes sans en être le descendant (*transferts horizontaux*). Un arbre phylogénétique

ne permet pas de représenter de tels transferts, c'est pourquoi on utilise parfois un *graphe phylogénétique*. Ces transferts horizontaux semblent surtout importants chez les bactéries – contrairement aux eucaryotes, leur matériel génétique n'est pas protégé par un noyau. Par exemple, la transduction est le processus par lequel de l'ADN est transféré entre des bactéries non parentes par l'intermédiaire d'un virus.

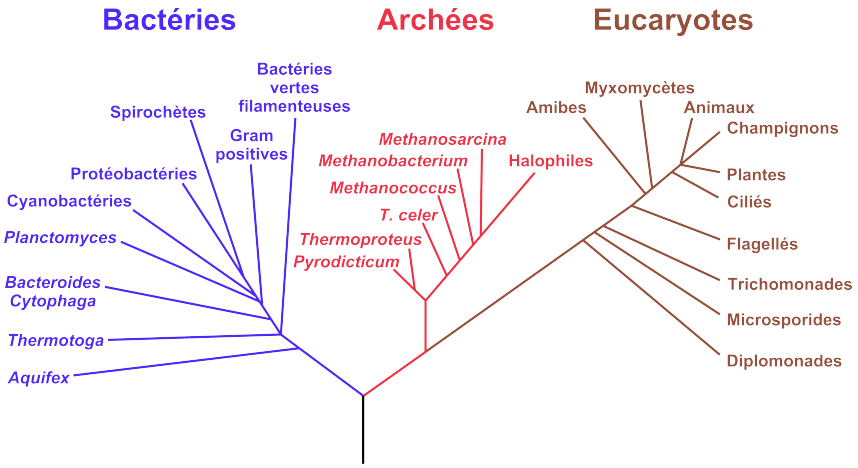


FIGURE 2.13 – Exemple d'arbre phylogénétique du monde vivant par [Mor03].

Avec ces mécanismes d'évolutions, les réseaux d'interactions entre protéines évoluent également. Berg *et al.* exposent deux processus principaux [BLW04].

Le premier fait suite à une mutation dans l'ADN (substitution, suppression ou insertion de nucléotide), pouvant être due à une erreur pendant sa copie, ou à un agent susceptible de provoquer des mutations (par exemple les rayons U.V.). Cette mutation peut entraîner la non-traduction de la protéine, ou modifier les interactions de la protéine codée – cette dernière peut alors perdre ou gagner des interactions.

Le second processus concerne la duplication de gène, entrainant la copie d'une protéine avec l'ensemble de ses interactions – le graphe s'agrandit. Selon Berg *et al.*, il s'avère cependant que les interactions redondantes dues à cette copie sont souvent perdues par la suite.

Plus des espèces sont proches dans l'arbre phylogénétique, plus elles sont susceptibles d'avoir des réseaux d'interactions entre protéines similaires, ainsi que des complexes protéiques en commun – notion importante pour la suite de notre étude.

On note que l'étude de réseaux métaboliques est également importante pour comprendre leur évolution. Nous n'entrons pas dans les détails, le lecteur peut se tourner vers l'article de Schmidt *et al.* proposant différents scénarios d'évolution du métabolisme [SSBD03].

Deuxième partie

Algorithmique des réseaux biologiques

Chapitre 3

Des problèmes algorithmiques liés aux réseaux biologiques

Contenu

Ce chapitre porte sur quelques problématiques liées aux réseaux biologiques, qui ont été présentés dans le chapitre précédent. C'est pourquoi nous donnons dans un premier temps quelques motivations biologiques concernant l'utilisation de réseaux biologiques, avant d'indiquer certaines questions algorithmiques sous-jacentes à ces motivations. Nous détaillons ensuite les algorithmes existants sur le problème de requête dans un réseau, en développant la présentation de l'algorithme (et du logiciel associé) que nous avons proposé.

3.1 Des motivations biologiques...

L'analyse de séquences est un aspect considérablement étudié en bioinformatique [Gus97, Pev06] afin d'améliorer notre connaissance d'entités biologiques comme l'ADN, l'ARN ou les protéines. Plus récemment, l'utilisation de réseaux biologiques (voir Section 2.2) a permis de répondre à de nouvelles problématiques, sur des objets non séquentiels. Nous citons ci-après quelques possibilités de leurs applications.

Utiliser des réseaux biologiques permet de déterminer quels groupes de protéines sont communs à plusieurs espèces [SI06]. Par conséquent, on peut envisager une meilleure caractérisation des fonctions d'espèces mal connues *via* ces similarités.

Des réseaux biologiques sont également utilisés pour mieux comprendre les infec-

tions et développer la lutte contre celles-ci [IS08]. En effet, lorsqu'un corps étranger pénètre dans l'organisme (par exemple le virus de l'herpès [UDZ+06]), il interagit avec celui-ci. En particulier, il pénètre dans le réseau d'interactions entre protéines. Un organisme malade aura donc des réseaux d'interactions entre protéines et de régulation de gènes différents de ceux d'un organisme sain. Par exemple, la réponse du système immunitaire engendre des modifications dans ces réseaux [CXR+05]. Comparer ces réseaux et identifier les différences peut permettre de mieux comprendre les effets de la maladie [CC08], ce qui participe au développement de traitements adaptés [CLR+06, IS08].

En outre, il y a de plus en plus de données disponibles concernant les réseaux d'agents infectieux. En comparant ces réseaux à ceux de l'homme, on peut identifier des interactions communes, mais aussi essayer de connaître les interactions à modifier afin de développer des traitements efficaces [IS08].

Enfin, l'utilisation de réseaux permet d'effectuer des simulations sur ordinateur plutôt que directement sur des organismes vivants. Ceci offre des avantages de coûts, mais aussi de reproductibilité des expériences ainsi qu'un passage à une plus large échelle [DVLMS06].

3.2 ...et les questions algorithmiques sous-jacentes

Sharan et Ideker [SI06] mettent en évidence trois grands types de problèmes algorithmiques issus des motivations vues précédemment (voir Tableau 3.1) : *l'alignement de plusieurs réseaux*, *l'intégration de réseaux* et *la recherche de sous-réseaux selon une requête*. Cette thèse s'intéresse particulièrement aux problèmes de recherche de sous-réseaux. Mais, nous verrons aussi dans le Chapitre 7 un travail portant sur la comparaison de réseaux de différents types, proche des problèmes d'intégrations.

3.2.1 L'alignement de plusieurs réseaux

Le premier type de problème concerne l'*alignement de plusieurs réseaux* d'un même type (on entend par type, l'information représentée par le réseau, comme le métabolisme ou les interactions entre protéines). On peut aussi dire qu'il s'agit de comparaison de réseaux homogènes. Dans la suite, on considère les réseaux comme des réseaux d'interactions entre protéines. Le problème consiste à superposer des sommets similaires, afin de pouvoir mettre en évidence des régions communes ou, au contraire, des régions différentes. Dans le cadre de réseaux d'interactions entre protéines, deux sommets sont dits similaires si les protéines qu'ils représentent sont homologues. Les algorithmes de la littérature présentés ci-après proposent très souvent des approches heuristiques pour résoudre ce type de problème.

Les réseaux alignés peuvent être d'espèces différentes, ou bien d'une même espèce mais acquis à des instants différents ou sous différentes conditions.

Problème	Entrées	Buts principaux
Alignement	Au moins deux réseaux de même type	Identification de groupes fonctionnels ; Étude de l'évolution ; Prédiction d'interactions
Intégration	Au moins deux réseaux de types différents d'une même espèce	Identification de groupes fonctionnels ; Étude des relations entre les données de différents types ; Prédiction d'interactions
Recherche de sous-réseaux	Un réseau requête et un réseau	Identification d'instances conservées ou dupliquées de la requête dans le réseau ; Étude des transferts

TABLE 3.1 – Synthèse des entrées et buts de trois problèmes concernant les réseaux biologiques, selon [SI06].

On peut considérer deux types d'algorithmes : ceux qui effectuent un alignement *global* des réseaux et ceux qui effectuent un alignement *local* (une telle distinction existe également dans les algorithmes plus anciens d'alignement de séquences). Dans le premier cas, l'ensemble des sommets des réseaux doit être aligné, pour maximiser un certain score de similarité. Parmi les algorithmes effectuant ce type d'alignement, citons celui de [ZBV09], où les auteurs formulent le problème sous forme de *matching* de graphes, afin d'utiliser des algorithmes connus, ou celui de [LLB+09], où deux sommets sont alignés s'ils sont similaires, mais aussi s'ils ont un voisinage similaire.

Dans le cas d'un alignement local, on cherche des sous-réseaux similaires conservés intéressants biologiquement. Deux types de structures sont principalement recherchées :

- les *voies de transduction du signal* (mécanisme par lequel une cellule répond à une information reçue), correspondant à des chemins dans le graphe,
- les *complexes protéiques* (ensemble dense de plusieurs protéines liées par des interactions), correspondant à des sous-graphes proches d'une clique.

Parmi les nombreux algorithmes proposant cette approche, la plupart utilisent un *graphe d'alignement*. Intuitivement, ce graphe regroupe les t réseaux à comparer. Il est en général construit de la manière suivante : chaque sommet contient un groupe de t protéines similaires, une pour chaque réseau. Une arête existe entre deux sommets de ce graphe d'alignement s'il existe des liens entre les protéines de chaque réseau (de la souplesse est souvent ajoutée). Ensuite, les structures intéressantes sont recherchées dans ce graphe d'alignement.

Les premiers algorithmes comme PathBLAST [KSK+03] ou MaWish [KGS05] ne permettaient l'alignement simultané que de deux réseaux – à cause de contraintes algorithmiques. Des algorithmes plus récents comme NetworkBLAST [SSK+05] ou

NetworkBLAST-M [KBS08] permettent l'alignement jusqu'à respectivement 3 et 10 réseaux.

3.2.2 L'intégration de plusieurs réseaux

L'*intégration de réseaux* consiste à combiner plusieurs réseaux de types différents construits avec le même ensemble de sommets. Si la définition semble proche du problème précédent, il est important de remarquer que les réseaux comparés ici sont de types différents, ils sont dits hétérogènes. Chaque type de réseau comporte une information différente d'une même cellule, chacun d'entre eux est une vue complémentaire de la cellule. Combiner, intégrer ces différents types de réseaux peut permettre une meilleure compréhension globale du fonctionnement de la cellule [SI06].

Une méthode courante consiste à utiliser un même ensemble de sommets pour les deux réseaux à intégrer, à donner des arêtes différentes selon le réseau, puis à chercher des sous-réseaux contenant les différents types d'arêtes afin de corroborer la présence d'interactions [SI06]. Plusieurs algorithmes proposent une réponse à ce type de problème, comme ceux décrits dans [LDAM04] ou [BML$^+$05].

3.2.3 La recherche de sous-réseaux

Ce problème consiste à rechercher un sous-réseau similaire à une requête donnée dans un réseau. On peut voir ce problème comme celui d'une requête dans une base de données, ou comme celui d'un motif dans un texte. Ici, on cherche à savoir si le réseau biologique contient une certaine requête (connue au préalable). La similarité entre la requête et la solution prend en compte les homologies (l'ensemble de sommets de la solution doit être identique à l'ensemble de sommets de la requête), mais la topologie de la requête doit également être conservée dans la solution. Nous verrons par la suite que cette topologie peut être celle d'un chemin, d'un arbre, voire de certains graphes.

Étant donnée l'imprécision des données biologiques, demander une solution exacte est généralement trop restrictif. En revanche, on peut chercher une solution la plus proche possible de la requête. On considère généralement deux types de flexibilités (voir aussi Figure 3.1) :

- des sommets demandés dans la requête sont absents de la solution (ce sont des *délétions*),
- des sommets présents dans la solution ne sont pas demandés dans la requête (ce sont des *insertions*).

Des contraintes algorithmiques limitent la taille des requêtes. Ces problèmes sont en effet proches des problèmes de morphismes de graphes, qui sont difficiles algorithmiquement.

Comme déjà précisé, le but de ce problème est de déterminer si des voies de transductions de signal ou des complexes protéiques connus (la requête) apparaissent dans

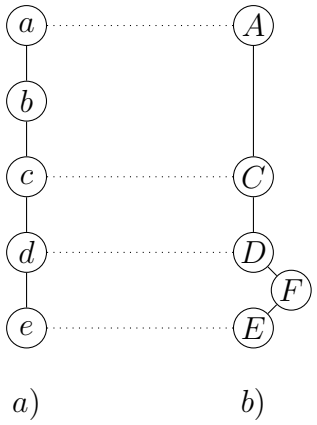


FIGURE 3.1 – Illustration du principe des insertions et des délétions. La requête est représentée en a), la solution extraite du réseau en b). On considère que deux lettres identiques représentent des homologues (par exemple, les sommets c et C sont homologues). b) est une solution pour cette requête, si deux insertions/délétions sont autorisées. En effet, le sommet b était demandé dans la requête mais est absent de la solution. De plus, le sommet F a été ajouté à la solution entre D et E.

le réseau. Ceci permet une meilleure connaissance de certaines espèces, en transférant l'information d'une espèce à une autre.

L'algorithme le plus ancien présenté ici, PathBLAST [KSK+03], ne permet des requêtes que sous formes de chemins. La complexité factorielle selon la taille du chemin limite en pratique la taille de ce dernier à 5 (selon [SSRS06]).

QPath [SSRS06] est un algorithme de complexité paramétrée, permettant la recherche de chemins. La complexité de cet algorithme est de $2^{\mathcal{O}(k+N_{ins})}mN_{del}$, où k est la taille de la requête, m le nombre d'arêtes dans le réseau, et N_{ins} (resp. N_{del}) le nombre d'insertions (resp. délétions) autorisées. Comme cette complexité est moins élevée que celle de PathBLAST grâce à l'utilisation de la technique du color-coding et de la programmation dynamique, les auteurs multiplient par deux la taille possible des requêtes en allant jusqu'à 10. Le « $\mathcal{O}$ » de la complexité présentée ci-dessus cache toutefois le nombre exponentiel en k ($\mathcal{O}(e^k)$) de relances de l'algorithme dues à l'utilisation de la technique du color-coding randomisé. L'algorithme est testé sur des données réelles avec plusieurs expériences. Parmi celles-ci, les auteurs utilisent QPath pour rechercher dans le réseau d'interactions entre protéines de la mouche *Drosophila melanogaster* des voies de signalisation connues (MAPK) de la levure *Saccharomyces cerevisiae*, symbolisées par des chemins composés de protéines. Il est annoncé que 63% de ces chemins sont retrouvés, avec au plus trois insertions ou délétions.

Une évolution dans le type de topologie de requêtes est donnée par Pinter *et al.* [PRYLZU05] avec MetaPathwayHunter, dans le cadre des réseaux métaboliques. En effet, leur algorithme permet des topologies d'arbres dans la requête. Cependant, le réseau doit alors être une forêt et non un graphe. La complexité de l'algorithme est alors polynomiale – le problème n'est pas NP-Complet car appliqué sur une forêt.

Une extension à QPath nommée QNet est proposée par Dost *et al.* [DSG+07]. Cet

algorithme accepte des requêtes sous forme d'arbres, en utilisant un algorithme similaire à QPath, alliant la programmation dynamique à la technique du color-coding. Étant donnée une coloration, chaque essai est de complexité similaire à QPath, c'est-à-dire de $2^{\mathcal{O}(k+N_{ins})}mN_{del}$, où k est la taille de la requête, m le nombre d'arêtes dans le réseau, et N_{ins} (resp. N_{del}) le nombre d'insertions (resp. délétions) autorisées. Un nombre exponentiel d'essais est requis en raison de l'utilisation de la technique du color-coding randomisé. Comme pour QPath, des expériences sur des données réelles sont proposées, consistant notamment à chercher dans le réseau d'interactions entre protéines de la mouche *Drosophila melanogaster* des voies de signalisations connues de la levure *Saccharomyces cerevisiae* et de l'homme (MAPK).

Dans un second temps, les auteurs de QNet expliquent que leur algorithme peut accepter des requêtes sous forme de graphes. Intuitivement, plus un graphe « ressemble » à un arbre, plus sa *treewidth* est petite. Un arbre a une treewidth de 1. La notion associée de décomposition arborescente consiste à disposer les sommets d'un graphe en groupes de sommets connectés dans un arbre – la treewidth mesure alors la taille du plus gros groupe de sommets. Une définition formelle peut se trouver par exemple dans [Die05] ou [Nie06]. Par rapport à l'algorithme gérant les arbres en tant que requête, la complexité en temps de chaque essai de QNet lorsque la requête est un graphe est multiplié par n^{t+1}, où t est la treewidth de la requête. Ainsi, QNet est toujours de complexité paramétrée (selon la taille de la requête) si le graphe requête est « proche » d'un arbre, s'il a une treewidth bornée. Les auteurs ne proposent pas d'implémentation de leur algorithme acceptant des graphes en tant que requête.

Enfin, avec Guillaume Blin et Stéphane Vialette, nous avons proposé PADA1, un algorithme permettant la requête de graphes dans un réseau. La version courte de ce travail a été publié dans [BSV09] et la version longue dans [BSV10b]. Tout comme QNet, nous autorisons les insertions et les délétions. Toujours comme QNet, nous transformons la requête en arbre avant d'effectuer la recherche de cet arbre dans le réseau par programmation dynamique. La principale différence entre les deux algorithmes se trouve dans la manière d'obtenir cet arbre. Alors que QNet utilise une décomposition arborescente, nous avons choisi de supprimer les arêtes de la requête créant des cycles en dupliquant des sommets. Par conséquent, QNet reste de complexité paramétrée si le graphe de requête est de treewidth bornée, alors que notre algorithme reste de complexité paramétrée si la requête à un *feedback vertex set* de taille bornée. Le feedback vertex set d'un graphe correspond au nombre de sommets à retirer dans un graphe pour retirer tous ses cycles.

Nous voulons obtenir un arbre depuis un graphe, sans perdre d'informations. C'est pourquoi nous dupliquons les sommets créant des cycles plutôt que de les supprimer – nous pouvons reconstruire le graphe depuis l'arbre contenant des sommets dupliqués. La technique de duplication des sommets est illustrée Figure 3.2.

Nous devons minimiser le nombre de sommets à dupliquer car ce nombre entre

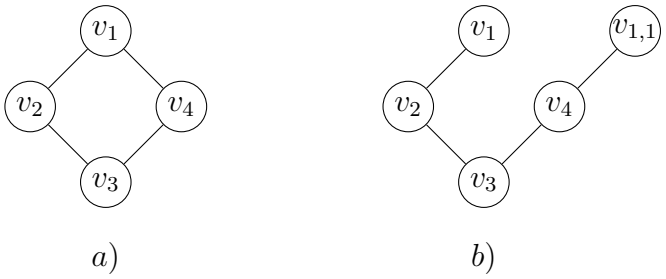


FIGURE 3.2 – Illustration du principe de duplication de sommets pour supprimer les cycles d'un graphe. En $a)$, le graphe contient un cycle. En $b)$, le sommet v_1 a été dupliqué afin de supprimer le cycle, en transformant l'arête $\{v_4, v_1\}$ en une arête $\{v_4, v_{1,1}\}$, utilisant le sommet $v_{1,1}$, dupliqué depuis le sommet v_1. Les sommets v_1 et $v_{1,1}$ représentent donc le même sommet v_1 dans le graphe de départ $a)$.

directement dans la complexité de notre algorithme – c'est pourquoi nous utilisons un algorithme donnant un feedback vertex set. Ce problème est NP-Complet [GJ79], cependant, il y a rarement plus d'une quinzaine de sommets dans les requêtes (nous sommes en effet limité par la complexité exponentielle en k, le nombre de sommets dans la requête, lors de la programmation dynamique). Nous pouvons donc utiliser un algorithme exponentiel exact afin de calculer ce feedback vertex set (on peut toutefois envisager une résolution par un algorithme efficace de complexité paramétrée utilisant la compression itérative comme [GGH+06], éventuellement couplé à l'obtention d'un noyau quadratique [Tho09]).

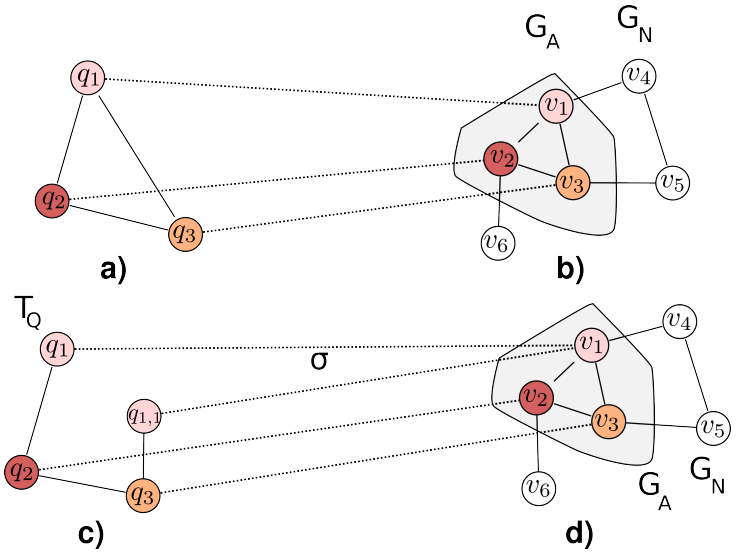


FIGURE 3.3 – Illustration de l'alignement effectué par PADA1. Le graphe requête de départ est en $a)$, il est transformé en arbre avec un sommet dupliqué, en $c)$. Le réseau G_N dans lequel est effectué la requête est présenté en $b)$ et $d)$. La fonction σ donne les sommets du réseau étant alignés avec les sommets de la requête. Comme q_1 et $q_{1,1}$ représentent le même sommet q_1 dans le graphe de départ, ils doivent être alignés avec le même sommet dans le réseau. Le sous-graphe du réseau correspondant à la requête est nommé G_A.

Par la suite, les sommets de l'arbre ne sont pas tous équivalents. En effet, deux

sommets dupliqués depuis le même sommet de départ ne doivent correspondre qu'à un seul sommet dans la solution car ils représentent la même protéine dans la requête (comme illustré Figure 3.3). Une fois le feedback vertex set F calculé, nous devons déterminer avec quels sommets du réseau ces sommets doivent être alignés : il y a n sommets possibles pour les $|F|$ sommets dupliqués. Ensuite, pour chacune de ces $n^{|F|}$ assignations possibles, nous effectuons l'algorithme de programmation dynamique recherchant un arbre dans un réseau. Cet algorithme doit cependant veiller à satisfaire ces contraintes (nous avons présenté ces récurrences dans [BSV10b]). On remarque que ce nombre de $n^{|F|}$ est une borne largement supérieure. Le nombre de sommets du réseau avec lequel nous tentons un alignement correspond en fait au nombre de sommets dont la protéine représentée est homologue avec la protéine représentée par le sommet dupliqué. Ce nombre est en pratique très faible (mais largement dépendant du seuil d'homologie fixée pour BLAST, algorithme permettant de déterminer les homologies).

Finalement, la complexité de l'algorithme est de $n^{|F|}.2^{\mathcal{O}(k+N_{ins})}mN_{del}$, où n et m sont respectivement le nombre de sommets et d'arcs dans le réseau, k est le nombre de sommets dans la requête, $|F|$ est le nombre de sommets du feedback vertex set de la requête, et N_{ins} et N_{del} sont respectivement le nombre d'insertions et de délétions autorisées. Notons que l'algorithme, comme QPath et QNet, utilise la technique du color-coding randomisé et doit donc être lancé un nombre exponentiel en k de fois – ceci est caché dans le $\mathcal{O}$ de la complexité. On remarque que la complexité dépend exclusivement du nombre de sommets à dupliquer (la taille du feedback vertex set) et non du nombre de duplications.

Enfin, précisons que nous avons fourni une implémentation python de PADA1, premier logiciel capable de rechercher des requêtes sous forme de graphes dans un réseau d'interactions entre protéines (QNet n'autorise pas les graphes dans leur logiciel, sans doute à cause de la difficulté inhérente à la décomposition arborescente). La Figure 3.4 illustre la recherche d'un chemin Mitogen-Activated Protein Kinase (MAPK) de l'homme dans le réseau d'interactions entre protéines de la mouche *Drosophila melanogaster*. Cet exemple de requête est repris depuis [DSG+07], où il est considéré d'importance.

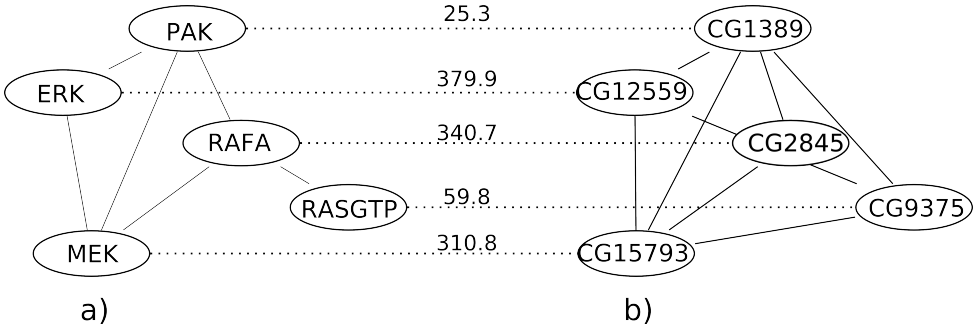


FIGURE 3.4 – Un exemple de production de notre algorithme. En a), une requête sous forme de graphe provenant d'une voie de signalisation de l'homme. En b), le résultat de l'alignement fourni par PADA1 dans le réseau d'interactions entre protéines de la mouche *Drosophila melanogaster*. Les lignes en pointillés représentent les homologies – nous avons également placé le score d'homologie entre les deux protéines, fourni par BLAST.

Chapitre 4

Recherche exacte de motifs dans des graphes colorés

Contenu

Dans ce chapitre, nous présentons et motivons le problème GRAPH MOTIF, largement étudié dans la suite de cette thèse. Il peut être considéré comme une variante des problèmes de recherche de sous-réseaux vus précédemment. Dans la suite du chapitre, nous nous intéressons à la version exacte du problème. Nous donnons dans un premier temps les résultats sur la complexité classique des problèmes. Comme ces problèmes sont difficiles même pour des instances très contraintes, une grande partie de la littérature est consacrée à la recherche d'algorithmes de complexité paramétrée – nous donnons les résultats connus, ainsi que nos contributions, consistant à une diminution de la complexité en temps et en espace de ces problèmes. Une courte section est ensuite consacrée à la recherche de noyau, avant de s'intéresser au problème de comptage que nous avons introduit. Nous montrons que ce dernier est dans la classe FPT, seulement si le motif est colorful.

4.1 Motivations et définition

Avant de motiver son utilisation dans les réseaux métaboliques ou les réseaux d'interactions entre protéines, nous donnons immédiatement sa définition [LFS06, FFHV07] ainsi qu'un exemple Figure 4.1.

EXACT GRAPH MOTIF

• **Entrée** : Un graphe $G = (V, E)$, un ensemble de couleurs C, une fonction $col : V \to C$, un multi-ensemble M de taille k sur C.

• **Sortie** : Un sous-ensemble $V_T \subseteq V$ tel que (i) $|V_T| = k$, (ii) $G[V_T]$ soit connexe et (iii) $col(V_T) = M$.

On appelle motif le multi-ensemble M. En toute généralité, le motif est considéré comme un multi-ensemble (plusieurs occurrences d'une couleur peuvent être demandées). La taille de M, notée $|M|$, est égale à la somme des multiplicités des couleurs de M, où la multiplicité d'une couleur c dans M est égale à son nombre d'occurrences dans M, noté $occ_M(c)$. Dans le cas spécifique où M est un ensemble simple, ce dernier est dit *colorful*.

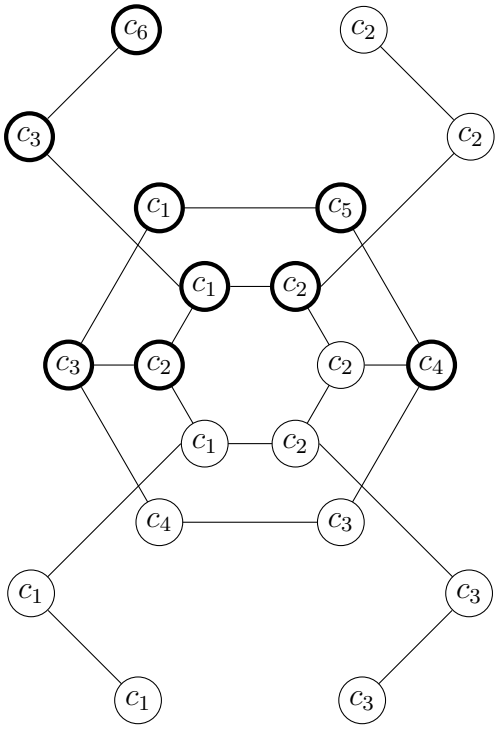


FIGURE 4.1 – Un exemple fictif d'un graphe G coloré sur les sommets par les couleurs $c_i, 1 \leq i \leq 6$. On trouve en gras dans le réseau une solution possible pour le problème EXACT GRAPH MOTIF avec le motif $M = \{c_1, c_1, c_2, c_2, c_3, c_3, c_4, c_5, c_6\}$.

Le problème est initialement introduit dans le cadre des réseaux métaboliques [LFS06]. Ce dernier est représenté par le graphe des réactions, comme présenté Section 2.2.2. Cependant, le graphe utilisé dans ce problème est non-orienté, contrairement

à la définition donnée plus tôt. En effet, selon [LFS06], on ne connait que rarement le sens des réactions – on trouve parfois même des contradictions dans une même base de données. Par conséquent, on considère ici que les réactions peuvent être réversibles, expliquant l'utilisation d'un graphe non-orienté.

Tous les sommets du réseau représentant des réactions ne sont pas considérés équivalents. Chaque réaction fait partie d'une classe fonctionnelle particulière. Lacroix *et al.* [LFS06] proposent deux méthodes possibles pour déterminer de telles classes, en ce basant sur l'enzyme de la réaction. La première consiste à comparer les enzymes catalysant les réactions. Si la similarité entre deux enzymes dépasse un certain score (par exemple la similarité de leur séquence d'acides aminés), alors elles ont une même couleur. La seconde méthode se base sur une classification des enzymes nommée *EC number* pour Enzyme Commission Number. Ce numéro indique la classe fonctionnelle de chaque enzyme (le type de réactions qu'elle catalyse telle que l'oxydo-réduction, l'hydrolyse...).

Dans ce problème, le motif correspond à un multi-ensemble de couleurs, donc à un multi-ensemble de fonctions demandées.

Contrairement aux problèmes de la section précédente, on n'impose pas une solution avec une topologie précise – la contrainte sur la topologie du motif est faible, c'est une information secondaire [LFS06]. Seule la connexité de la solution est requise. Le point de vue adopté est fonctionnel, c'est la fonction des éléments qui porte la fonction de la solution et non sa topologie. En effet, des complexes de topologies similaires peuvent représenter des fonctions très diverses [LFS06].

Un peu plus tard, le problème GRAPH MOTIF est adapté dans le cadre des réseaux d'interactions entre protéines [BHK+09]. Le graphe représenté, les couleurs ainsi que les motivations sont légèrement différents par rapport aux réseaux métaboliques.

Le graphe représente le réseau d'interactions entre les protéines (comme présenté Section 2.2.2). Un motif est un complexe protéique, ou encore une voie de transduction du signal [BHK+09].

Une couleur distincte est donnée à chaque protéine du motif – un motif de taille k contient donc k couleurs différentes. On colore le réseau de la manière suivante : un sommet représentant une protéine p_1 est coloré avec la couleur d'une protéine p_2 présente dans le motif si p_1 et p_2 sont homologues.

On remarque que la méthode précédente crée uniquement des motifs colorful. Selon Vincent Lacroix (discussion avec l'auteur), deux approches sont envisageables pour obtenir des motifs contenant plusieurs occurrences de chaque couleur. La première est une approche fonctionnelle utilisant une classification telle que Gene Ontology [ABB+00]. Deux protéines du motif possèdent la même couleur si elles appartiennent à la même classe. La seconde est une approche évolutive utilisant les similarités entre protéines. Deux protéines du motif possèdent la même couleur si elles sont homologues.

À l'heure actuelle, l'information sur la topologie des motifs recherchés dans le cadre de réseaux d'interactions entre protéines est souvent manquante [BHK+09]. De plus, il existe beaucoup de faux-positifs et de faux-négatifs dans de tels réseaux. Par conséquent, demander une topologie précise comme un chemin, un arbre ou un graphe semble parfois non pertinent. Une contrainte sur la topologie plus faible comme la simple connexité paraît donc justifiée [BHK+09, Lac07].

Il est également noté dans Betzler *et al.* [BFKN08] que le problème est applicable à des réseaux techniques ou sociaux. Les auteurs ne donnent pas plus de détails. Néanmoins, une discussion avec Christophe Prieur nous permet de donner quelques idées d'applications de ce problème dans ce cadre.
Obtenir un échantillon représentatif d'un grand réseau est un problème complexe, étudié depuis de nombreuses années [Gra76]. On peut imaginer un graphe où les sommets représentent des individus et les arêtes leurs relations. Des couleurs sur les sommets peuvent permettre de donner des informations supplémentaires sur les individus (par exemple leur tranche d'âge, leur sexe, leur statut institutionnel...). Alors, une solution au problème GRAPH MOTIF peut s'avérer intéressante pour l'élaboration d'un échantillonnage représentatif de l'ensemble du réseau.
On peut également imaginer une application pour ce problème dans le cadre de la création d'une équipe pour un projet (problème très étudié, comme par exemple dans [MWK08]). Le graphe représente alors des individus, il existe une arête entre deux sommets si les individus se connaissent, et les couleurs représentent les compétences des individus. Alors, si le motif représente les demandes pour le projet (par exemple, le projet nécessite deux webmasters, un commercial, trois développeurs...), une solution au problème donne une équipe possible respectant les compétences requises. De plus, comme la solution doit être connexe, chaque individu choisi dans la solution connaît au moins une autre personne de l'équipe.

4.2 Complexité classique

D'après la littérature, le problème EXACT GRAPH MOTIF s'avère rapidement difficile au regard de la complexité classique. En effet, ce problème est NP-Complet, même lorsque le motif est colorful et G est un arbre enraciné de hauteur 2 [ABRH+10], ou un arbre de degré 3 [FFHV07] (Table 4.1).

Les résultats de cette table sont donnés quand le motif est colorful et sont donc valables quand le motif est un multi-ensemble (puisque le problème EXACT GRAPH MOTIF quand le motif est colorful est un cas particulier du problème où le motif est un multi-ensemble). Dans le cas multi-ensemble, le problème reste NP-Complet si le

graphe est biparti de degré maximum quatre et si seulement deux couleurs sont utilisées [FFHV07] (le problème est par contre polynomial si un nombre constant de couleur est utilisé et si G est de treewidth bornée [FFHV07]).

Les algorithmes polynomiaux sont donnés quand le motif est colorful. On remarque toutefois que l'algorithme donné par [ABRH$^+$10] reste valide avec un multi-ensemble. Comme la solution doit être connexe, elle correspond donc forcement à un sous-chemin de G. On peut énumérer en temps polynomial tous les sous-chemins de taille $|M|$ et vérifier si l'ensemble des couleurs de ce sous-chemin est égal à M. Dans le cas de *caterpillars* (arbre où la suppression de toutes les feuilles donne un chemin), l'argument est similaire en testant en plus l'ajout des feuilles au sous-chemin.

Chemins	Caterpillars	Arbres de degré 3 ou enracinés de hauteur 2	Graphes
P [ABRH$^+$10]		NP-Complet [FFHV07, ABRH$^+$10]	

TABLE 4.1 – Synthèse du comportement en complexité classique du problème EXACT GRAPH MOTIF.

Enfin, on voit assez simplement que le problème EXACT GRAPH MOTIF est dans la classe P dans le cas spécifique où le motif est colorful, les couleurs n'apparaissent qu'au plus deux fois dans G et que G est un arbre.

Proposition 4.2.1. *Si G est un arbre dans lequel chaque couleur de C n'apparaît que deux fois au maximum et que le motif est colorful, alors le problème* EXACT GRAPH MOTIF *peut se résoudre avec un algorithme polynomial.*

Démonstration. On propose dans un premier temps de considérer un cas particulier du problème, où G est enraciné en r et où r est contenu dans une solution. En effet, s'il existe un algorithme polynomial pour ce cas particulier, alors on peut trouver un algorithme polynomial en lançant l'algorithme pour les $|V|$ racines possibles.

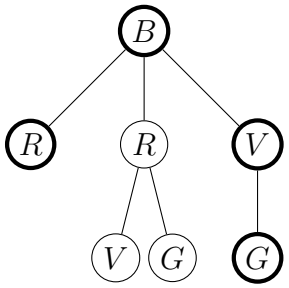
On propose de construire depuis une instance $\mathcal{I}$ du cas particulier exprimé ci-dessus, une instance $\mathcal{I}'$ du problème 2-SAT (cas particulier du problème SAT où chaque clause est de taille 2). Il est connu que 2-SAT peut se résoudre en temps polynomial [GJ79].

Selon un parcours donné (disons préfixe) de l'arbre $G = (V, E)$ de $\mathcal{I}$, on note pour chaque sommet $v \in V$, $a_v = x_{col(v)}$ si v est la première occurrence de $col(v)$ dans G, et $a_v = \neg x_{col(v)}$ si v est la seconde occurrence de $col(v)$ dans G. Chaque sommet $v \in V$ (exceptée la racine r) possède un unique père, noté $p(v)$. On propose alors l'ensemble des clauses $\{(\neg a_v \vee a_{p(v)}) : v \in V \setminus \{r\}\}$. Ceci symbolise, pour chaque $v \in V$, la formule logique $(a_v \Rightarrow a_{p(v)})$, indiquant que si a_v est vrai, alors $a_{p(v)}$ doit l'être aussi. On précise que l'ensemble des variables pour $\mathcal{I}'$ est $X = \{x_c : c \in C\}$ (voir Figure 4.2 pour un exemple de construction).

Nous voyons maintenant qu'il existe une solution colorful dans G pour $\mathcal{I}$ si et seulement si il existe une affectation des variables rendant vraies toutes les clauses

de $\mathcal{I}'$. Étant donnée une solution colorful $T = (V_T, E_T)$ pour $\mathcal{I}$, on place $a_v = Vrai$ si $v \in V_T$, $a_v = Faux$ sinon. Par conséquent, chaque variable $x_c \in X$ possède une unique affectation. On voit également que toutes les clauses sont vraies. En effet, pour chaque arête $\{v, p(v)\} \in E$, la clause $(\neg a_v \vee a_{p(v)})$ est vraie sauf si $v \in V_T$ et $p(v) \notin V_T$. Cependant, cette configuration est impossible dans une solution contenant la racine.

Inversement, étant donnée une affectation des variables rendant vrai l'ensemble des clauses de $\mathcal{I}'$, on construit la solution $V_T = \{v : a_v = Vrai\}$. Chaque variable $x_c \in X$ possède une unique affectation. Par conséquent, il existe exactement une occurrence de la couleur c dans V_T. Enfin, comme toutes les clauses sont vraies, $G[V_T]$ est connexe. En effet, $\forall v \in V$, si la clause $(\neg a_v \vee a_{p(v)})$ est vraie, alors il est impossible que $a_v = Vrai$ et $a_{p(v)} = Faux$, seule configuration possible ne donnant pas une solution connexe. □



$G = (V, E)$

FIGURE 4.2 – Depuis le graphe $G = (V, E)$ (dont les couleurs sont données dans les sommets), on construit l'ensemble de clauses suivant :$\{(\neg x_R \vee x_B), (\neg\neg x_R \vee x_B), (\neg x_V \vee \neg x_R), (\neg x_G \vee \neg x_R), (\neg\neg x_V \vee x_B), (\neg\neg x_G \vee \neg x_V)\}$. Une solution possible (en gras), correspond à l'affectation $x_R = x_B = Vrai$, $x_V = x_G = Faux$.

4.3 Complexité paramétrée

Comme le problème de décision EXACT GRAPH MOTIF est très rapidement NP-Complet, il est naturel de se tourner vers la complexité paramétrée pour espérer pouvoir le résoudre dans un temps raisonnable. L'article introduisant ce problème propose un algorithme de complexité paramétrée [LFS06]. Cependant, celui-ci n'est valable que si G est un arbre. Rapidement, plusieurs groupes ont cherché à améliorer cet algorithme. La Table 4.2 donne les meilleurs résultats en matière de complexité paramétrée, avant les résultats présentés dans cette thèse.

Si pour la complexité classique l'élément semblant donner la difficulté est la topologie de G, dans le cadre de la complexité paramétrée en prenant comme paramètre la taille de M (donc la taille de la solution), on trouve des résultats positifs dans tous les cas. Cependant, demander en paramètre le nombre de couleurs différentes rend le problème difficile d'un point de vue de la complexité paramétrée, même lorsque G est un arbre (réduction depuis le problème W[1]-Difficile CLIQUE) [FFHV07].

Motif \ Paramètre	k	c
colorful	$\mathcal{O}(3^k\|E\|)$ [BFKN08]	$\mathcal{O}(3^c\|E\|)$ [BFKN08]
multi-ensemble	$\mathcal{O}^*(4.32^k)$ [BFKN08]	W[1]-Difficile [FFHV07]

TABLE 4.2 – Étude de la complexité paramétrée (sans prendre en compte les contributions de cette thèse) pour le problème EXACT GRAPH MOTIF, en fonction du paramètre k (taille de M donc de la solution) et du paramètre c (nombre de couleurs différentes dans le motif M). Les complexités en espace sont exponentielles, tandis que le résultat de difficulté est également valide pour les arbres [FFHV07].

4.3.1 Avec un motif colorful

4.3.1.1 État de l'art

L'idée principale présentée dans [BFKN08] pour obtenir un algorithme de complexité $\mathcal{O}(3^k|E|)$, lorsque le motif est un ensemble simple, est assez proche de celle vue Section 1.3.2.1 pour chercher un chemin dans un graphe. On se base ici sur le fait que la solution doit être connexe. Par conséquent, il est possible de seulement chercher un arbre couvrant avec k sommets. En outre, comme le motif est colorful, la solution contient k couleurs différentes.

Par programmation dynamique, on indique qu'il existe un arbre colorful enraciné en v utilisant un ensemble S de couleurs s'il existe un voisin u de v et un sous-ensemble $S' \subseteq S$ tels qu'il existe un sous-arbre colorful de G enraciné en u utilisant un ensemble $S \setminus S'$ de couleurs et un sous-arbre colorful enraciné en v utilisant un ensemble S' de couleurs (voir Figure 4.3). En testant pour chaque voisin de v et chaque sous-ensemble possible $S' \subseteq S$ de couleurs (il y en a au pire $2^{|S|} \leq 2^k$), on calcule toutes les solutions possibles. Le cas de base est le suivant : on trouve un sous-arbre colorful enraciné en v utilisant un ensemble S de couleurs si $S = \{col(v)\}$.

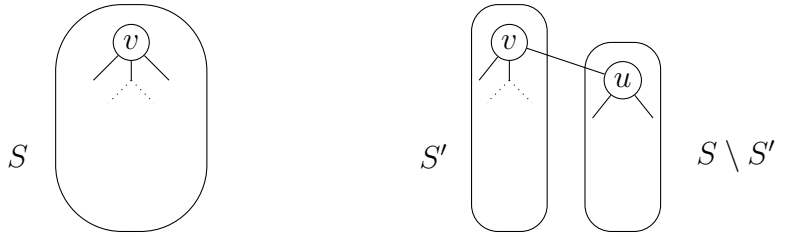


FIGURE 4.3 – Illustration de la programmation dynamique pour le problème EXACT GRAPH MOTIF quand le motif est colorful. Il existe un sous-arbre de G enraciné en v colorful sur un ensemble de couleurs S s'il existe un voisin u de v et un sous-ensemble $S' \subseteq S$ tels qu'il existe un sous-arbre de G enraciné en u colorful sur $S \setminus S'$ et un sous-arbre de G enraciné en v colorful sur S'.

Bruckner *et al.* [BHK+09] remarquent qu'un algorithme de complexité $\mathcal{O}^*(2^k)$ est possible mais ne donnent pas les détails de son obtention.

4.3.1.2 Quand EXACT GRAPH MOTIF avec motif colorful rencontre les monômes

Cette section reprend une partie de l'article effectué en collaboration avec Sylvain Guillemot, publié dans [GS10]. Notamment, on montre comment l'utilisation de la technique de détection de monômes multilinéaires *via* le problème MULTILINEAR DETECTION vu Section 1.3.2.2 permet de diminuer la complexité en temps du problème EXACT GRAPH MOTIF quand le motif est colorful à $\mathcal{O}^*(2^k)$ et avec une complexité en espace polynomiale. La solution au problème EXACT GRAPH MOTIF devant être connexe, on peut se ramener à un problème cherchant un arbre couvrant $T = (V_T, E_T)$ de la solution.

Proposition 4.3.1. *On peut résoudre le problème* EXACT GRAPH MOTIF *quand le motif est colorful par un algorithme randomisé avec une complexité en temps de $\tilde{\mathcal{O}}(2^k k^2|E|)$ et une complexité en espace de $\tilde{\mathcal{O}}(k^2|E|)$* [1].

Démonstration. Comme le motif est colorful et que la solution doit être connexe, on considère maintenant qu'une solution $T = (V_T, E_T)$ est un arbre, comportant une occurrence des $|C| = k$ couleurs apparaissant dans le graphe. Soit (G, k) une instance du problème EXACT GRAPH MOTIF. On introduit l'ensemble de variables suivant : $\{x_c : c \in C\}$, c'est-à-dire une variable pour chaque couleur. On construit un circuit arithmétique A contenant les sommets internes $P_{i,u}$ pour $1 \leq i \leq k, u \in V$. De manière informelle, les monômes multilinéaires de $P_{i,u}$ correspondent aux sous-arbres colorful de G ayant i sommets, dont u. La construction des sommets internes est comme suit, où $N(u)$ représente l'ensemble des voisins de u dans G :

$$P_{1,u} = x_{col(u)},$$
$$P_{i,u} = \sum_{i'=1}^{i-1} \sum_{v \in N(u)} P_{i',u} P_{i-i',v} \text{ si } i > 1.$$

On définit également $P = \sum_{u \in V} P_{k,u}$ comme étant la racine de A.

L'instance construite pour le problème MULTILINEAR DETECTION est (A, k). En appliquant le Théorème 1.3.1 sur cette instance et en observant que $|A| = \mathcal{O}(k^2|E|)$, on donne une solution à (A, k) en temps $\tilde{\mathcal{O}}(2^k k^2|E|)$, en utilisant $\tilde{\mathcal{O}}(k^2|E|)$ d'espace.

Il reste donc à montrer que cette réduction est correcte. Étant donné un ensemble $S \subseteq C$, on définit le monôme multilinéaire $\pi_S = \prod_{c \in S} x_c$. Pour un $u \in V$ et un $S \subseteq C$, une (u, S)-*solution* est un sous-arbre $T = (V_T, E_T)$ de G, tel que $u \in V_T$, T est coloré distinctement par col, et $col(V_T) = S$. On montre par induction sur $1 \leq i \leq k$ que π_S est un monôme multilinéaire de $P_{i,u}$ si et seulement si (i) $|S| = i$ et (ii) il existe une (u, S)-solution. Ceci est clairement vrai pour $i = 1$; supposons maintenant que $i \geq 2$ et que la propriété soit vraie pour chaque $1 \leq j < i$.

1. On rappelle que la notation $\tilde{\mathcal{O}}$ supprime les facteurs polylogarithmiques.

Supposons que $|S| = i$ et que $T = (V_T, E_T)$ est une (u, S)-solution, montrons que π_S est un monôme multilinéaire de $P_{i,u}$. Soit v un voisin de u dans T. Alors, la suppression de l'arête $\{u, v\}$ dans T produit deux arbres T_1 et T_2, où T_1 contient u et T_2 contient v. Ces deux arbres T_1 et T_2, de taille respective i_1 et i_2, sont respectivement colorés par les ensembles S_1 et S_2. Comme T_1 est une (u, S_1)-solution, π_{S_1} est un monôme multilinéaire de $P_{i_1,u}$ par l'hypothèse d'induction. Comme T_2 est une (v, S_2)-solution, π_{S_2} est un monôme multilinéaire de $P_{i_2,v}$ par l'hypothèse d'induction. Alors, $\pi_S = \pi_{S_1}\pi_{S_2}$ est un monôme multilinéaire de $P_{i_1,u}P_{i_2,v}$ et donc de $P_{i,u}$.

Inversement, supposons que π_S est un monôme multilinéaire de $P_{i,u}$. Par définition de $P_{i,u}$, il existe un $1 \leq i' \leq i-1$ et un $v \in N(u)$ tels que π_S est un monôme multilinéaire de $P_{i',u}P_{i-i',v}$. On peut alors donner une partition de S en deux ensembles S_1 et S_2, où π_{S_1} est un monôme multilinéaire de $P_{i',u}$ et π_{S_2} est un monôme multilinéaire de $P_{i-i',v}$. L'hypothèse d'induction implique alors que (i) $|S_1| = i'$ et $|S_2| = i - i'$, (ii) il existe une (u, S_1)-solution pour $T_1 = (V_1, E_1)$ et une (v, S_2)-solution pour $T_2 = (V_2, E_2)$. Comme S_1 et S_2 sont disjoints, alors $|S| = i$, ce qui prouve le point (i) ; de plus, V_1 et V_2 sont disjoints, par conséquent $T = (V_1 \cup V_2, E_1 \cup E_2 \cup \{u, v\})$ est une (u, S)-solution, ce qui prouve le point (ii). □

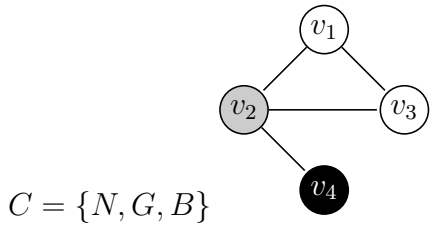


FIGURE 4.4 – Un exemple d'une instance pour EXACT GRAPH MOTIF. L'ensemble de couleurs est C, où la couleur noire est notée N, la grise G et la blanche B.

Pour illustrer cette construction, voyons par exemple une partie du circuit arithmétique pour le graphe présenté Figure 4.4. On voit que les monômes de degré $i = 3$ contenant le sommet v_1 sont l'ensemble des produits de monômes de degré $i' < i$ contenant le sommet 1 et ceux de degré $i - i'$ contenant un voisin de v_1.

$$
\begin{aligned}
P_{3,v_1} = \;& P_{1,v_1}P_{2,v_2} + P_{1,v_1}P_{2,v_3} + P_{2,v_1}P_{1,v_2} + P_{2,v_1}P_{1,v_3} \\
= \;& x_B(P_{2,v_2} + P_{2,v_3}) + P_{2,v_1}(x_G + x_B) \\
= \;& x_B(x_G(P_{1,v_1} + P_{1,v_3} + P_{1,v_4})) + x_B(P_{1,v_1} + P_{1,v_2}) + (x_B(P_{1,v_2} + P_{1,v_3}))(x_G + x_B) \\
= \;& x_B(x_G(x_B + x_B + x_N) + x_B(x_B + x_G)) + (x_B(x_G + x_B))(x_G + x_B) \\
= \;& x_Bx_Gx_B + x_Bx_Gx_B + x_Bx_Gx_N + x_Bx_Bx_B + x_Bx_Bx_G + x_Bx_Gx_G + \\
& x_Bx_Gx_B + x_Bx_Bx_G + x_Bx_Bx_B
\end{aligned}
$$

Il existe un monôme multilinéaire de degré $i = 3$ incluant le sommet $v_1 : x_B x_G x_N$.

4.3.2 Avec multi-ensemble comme motif

On considère dans cette section la variante du problème EXACT GRAPH MOTIF où le motif est un multi-ensemble de couleurs.

Un *matroïde* $\mathcal{M} = (S, \mathcal{I})$ (voir par exemple [Fes09]) est défini sur un ensemble fini $S = \{s_1, s_2, \ldots, s_{|S|}\}$ et un ensemble de sous-ensembles $\mathcal{I}$ sur S appelés *indépendants*. Les deux conditions suivantes doivent être satisfaites pour que $\mathcal{M}$ soit un matroïde :

1. si $X \subseteq Y$ et $Y \in \mathcal{I}$, alors $X \in \mathcal{I}$ (tout sous-ensemble d'un indépendant est un indépendant),
2. si $X, Y \in \mathcal{I}$ et $|Y| > |X|$, alors $\exists s \in Y \setminus X$ tel que $X \cup \{s\} \in \mathcal{I}$ (il existe un élément de Y qui n'est pas dans X qui peut être ajouté à X tel que X reste indépendant).

Un exemple de matroïde est le *matroïde de partition*, où il existe une partition de S en k ensembles (disjoints) $\{S_1, S_2, \ldots, S_k\}$, k entiers positifs $n_1, n_2, \ldots, n_k$ et $\mathcal{I} = \{X \subseteq S : |X \cap S_i| \leq n_i, \forall 1 \leq i \leq k\}$. Tout indépendant du matroïde de partition ne contient donc pas plus de n_i éléments de S_i.
Depuis une instance $(M, G = (V, E))$ du problème EXACT GRAPH MOTIF, on construit un matroïde de partition, où l'ensemble S est $S = \{v_i : v_i \in V\}$ et comporte une partition sur les couleurs en c ensembles (c est le nombre de couleurs différentes dans M) $S_i = \{v_j : col(v_j) = i\}, \forall 1 \leq i \leq c$. Les c entiers positifs correspondent aux occurrences de chaque couleur, $n_i = occ_M(i), 1 \leq i \leq c$. Un indépendant de ce matroïde de partition de taille M correspondant à un sous-graphe connexe dans G est donc une solution pour $(M, G = (V, E))$.
Est-ce que la généralisation de ce problème avec en entrée un matroïde quelconque au lieu d'un matroïde de partition reste dans la classe FPT ?

4.3.2.1 État de l'art

Si le premier algorithme de complexité paramétrée pour EXACT GRAPH MOTIF quand le motif est un multi-ensemble est présenté par [LFS06], celui-ci n'est valide que pour les arbres. Or, les réseaux considérés sont rarement des arbres mais plutôt des graphes. Fellows *et al.* [FFHV07] présentent un algorithme de complexité $\mathcal{O}^*(87^k)$ (selon [BFKN08]) lorsque le réseau est un graphe, en utilisant la technique du color-coding.

L'algorithme de [BFKN08] (dont la complexité est donnée Table 4.2) utilise l'algorithme pour EXACT GRAPH MOTIF quand le motif est colorful de [BFKN08] décrit Section 4.3.1.1. La technique du color-coding est utilisée quand le motif est un multi-ensemble. En effet, dans le cas colorful, nous sommes assurés que S' et $S \setminus S'$ sont

distincts car l'ensemble de couleurs de départ est colorful – ce n'est plus le cas si M est un multi-ensemble.

Cette fois-ci, l'idée est la suivante. Pour chaque couleur c_i dont le nombre d'occurrences dans le motif $occ_M(c_i)$ est supérieur à 1, on crée $occ_M(c_i)$ nouvelles couleurs $c_i^1, c_i^2, \ldots, c_i^{occ_M(c_i)}$, que l'on place dans le motif à la place de c_i. Ensuite, dans G, on recolore les sommets anciennement colorés par c_i par une couleur choisie aléatoirement parmi les $occ_M(c_i)$ nouvelles couleurs. Alors, le motif est colorful et le réseau est colorié avec les couleurs présentes dans le motif – on peut donc lancer l'algorithme vu précédemment. Cependant, il est possible que la coloration aléatoire ne permette pas de trouver une solution qui était pourtant présente dans l'instance de départ. C'est pourquoi il faut relancer cette coloration un nombre exponentiel de fois, comme pour l'algorithme utilisant la technique du color-coding pour la recherche d'un chemin simple contenant k sommets (Figure 4.5).

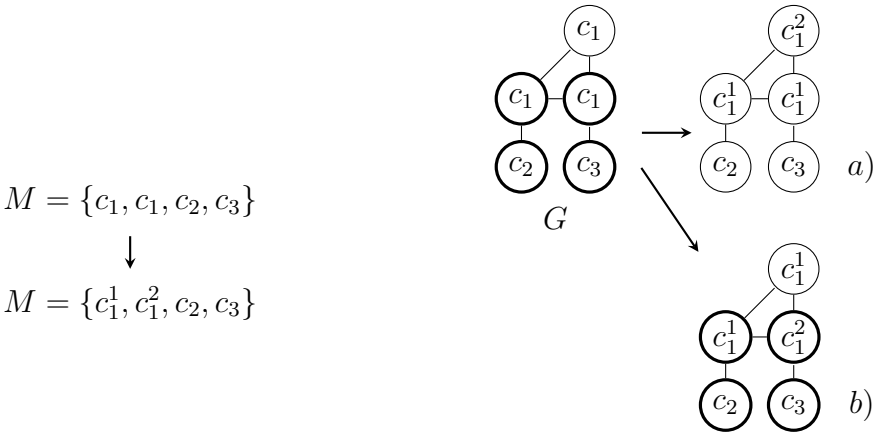


FIGURE 4.5 – Illustration de l'algorithme proposé par [BFKN08] pour le problème EXACT GRAPH MOTIF quand le motif est un multi-ensemble. La couleur c_1, apparaissant deux fois dans le motif, est remplacée par deux nouvelles couleurs c_1^1 et c_1^2. Dans le réseau, chaque sommet initialement coloré par c_1 obtient aléatoirement la couleur c_1^1 ou c_1^2. La première coloration aléatoire (en a)) n'accepte pas de solution. Pourtant, une solution existe dans le graphe original (en gras). La seconde coloration, en b) accepte, elle, une solution.

Les auteurs de [BFKN08] utilisent la technique *fast subset convolution* [BHKK07] pour diminuer la complexité de leur algorithme lors de la programmation dynamique. Cependant, nous ne savons pas si cette technique étudiée seulement en théorie peut être utilisée en pratique.

La complexité de l'algorithme par l'utilisation de la technique du color-coding est donc normalement de $\mathcal{O}^*(e^k 3^k) = \mathcal{O}^*(8.1^k)$. Toutefois, les auteurs utilisent deux techniques, (i) l'utilisation de plus de couleurs [HWZ08] et (ii) la technique de *fast subset*

convolution [BHKK07], permettant l'accélération de plusieurs algorithmes de programmation dynamique. Ces deux techniques entraînent la diminution de la complexité pire cas de l'algorithme à $\mathcal{O}^*(4.32^k)$. Notons que par l'utilisation de programmation dynamique, la complexité en espace est exponentielle.

4.3.2.2 Quand EXACT GRAPH MOTIF avec un multi-ensemble pour motif rencontre les monômes

Cette section reprend également une partie de l'article [GS10]. Comme précédemment, l'utilisation de la technique de détection de monômes multilinéaires *via* le problème MULTILINEAR DETECTION vu Section 1.3.2.2 permet de diminuer la complexité en temps du problème EXACT GRAPH MOTIF quand le motif est un multi-ensemble à $\mathcal{O}^*(4^k)$ et à une complexité en espace polynomiale (alors que les algorithmes déjà connus utilisent un espace exponentiel).

Proposition 4.3.2. *On peut résoudre le problème* EXACT GRAPH MOTIF *quand le motif est un multi-ensemble par un algorithme randomisé avec une complexité en temps de $\tilde{\mathcal{O}}(4^k k^2|E|)$ et une complexité en espace de $\tilde{\mathcal{O}}(k^2|E|)$.*

Démonstration. Pour chaque couleur $c \in C$ où $occ_M(c) = m$, on introduit les variables $y_{c,1}, ..., y_{c,m}$, ainsi qu'un sommet du circuit arithmétique $Q_c = y_{c,1} + ... + y_{c,m}$. Pour chaque sommet $u \in V$, on introduit une variable x_u. Le circuit est construit comme suit :

$$P_{1,u} = x_u Q_{col(u)},$$
$$P_{i,u} = \sum_{i'=1}^{i-1} \sum_{v \in N(u)} P_{i',u} P_{i-i',v} \text{ si } i > 1,$$

et $P = \sum_{u \in V} P_{k,u}$ est la racine du circuit A. On note que seul les sommets $P_{1,u}$ sont modifiés par rapport au circuit de la Proposition 4.3.1.

Intuitivement, les variables x_u forcent à choisir des sommets différents lors de la construction de l'arbre, tandis que les variables $y_{c,i}$ forcent qu'une couleur donnée n'apparaisse pas plus que demandée.

L'instance construite pour le problème MULTILINEAR DETECTION est $(A, 2k)$. On cherche en effet un monôme de degré $2k$ car deux variables représentent chaque sommet u de la solution (la variable x_u et la variable $y_{col(u),i}$ représentant l'occurrence de sa couleur). En appliquant le Théorème 1.3.1 sur cette instance et en observant que $|A| = \mathcal{O}(k^2|E|)$, on donne une solution à $(A, 2k)$ en temps $\tilde{\mathcal{O}}(4^k k^2|E|)$, en utilisant $\tilde{\mathcal{O}}(k^2|E|)$ d'espace.

Montrons maintenant que cette réduction est correcte. Pour cela, prouvons par induction sur $1 \leq i \leq k$ qu'il existe un monôme multilinéaire de la forme $x_{v_1} y_{c_1,j_1} ... x_{v_i} y_{c_i,j_i}$ dans $P_{i,u}$ si et seulement si il existe un arbre $T = (V_T, E_T)$ dans G tel que (i) $u \in V_T$,

(ii) $V_T = \{v_1, ..., v_i\}$ et (iii) $col(V_T) \subseteq M$. Ceci est clairement vrai pour $i = 1$. Supposons maintenant $i \geq 2$ et que la propriété soit vraie pour chaque $1 \leq i' < i$.

Supposons qu'il existe un arbre $T = (V_T, E_T)$ tel que (i) $u \in V_T$, (ii) $V_T = \{v_1, ..., v_i\}$ et (iii) $col(V_T) \subseteq M$ et montrons qu'il existe un monôme multilinéaire dans $P_{i,u}$. Soit v un voisin de u. En supprimant $\{u, v\} \in E_T$, nous obtenons deux arbres $T_1 = (V_{T_1}, E_{T_1})$ et $T_2 = (V_{T_2}, E_{T_2})$ tels que (i) $u \in V_{T_1}$ et $v \in V_{T_2}$, (ii) $V_{T_1} = \{v_1, ..., v_{i_1}\}$ et $V_{T_2} = \{v_1, ..., v_{i_2}\}$ et (iii) $col(V_{T_1}) \subseteq M$ et $col(V_{T_2}) \subseteq M$. Par l'hypothèse d'induction, il existe donc un monôme multilinéaire $\pi_{T_1} = \prod_{i'=1}^{i_1} x_{v_{i'}} y_{c_{i'},j_{i'}}$ dans $P_{i_1,u}$, où $v_{i'}$ est le i'^{eme} sommet de T_1, $c_{i'}$ est la couleur de $v_{i'}$ et $j_{i'}$ est le nombre d'occurrence de $c_{i'}$ dans $T_1[v_1, ..., v_{i'}]$. Toujours par l'hypothèse d'induction, il existe également un monôme multilinéaire $\pi_{T_2} = \prod_{i'=1}^{i_2} x_{v_{i'}} y_{c_{i'},j_{i'}}$ dans $P_{i_2,v}$, où $v_{i'}$ est le i'^{eme} sommet de T_2, $c_{i'}$ est la couleur de $v_{i'}$ et $j_{i'}$ est le nombre d'occurrence de $c_{i'}$ dans $T_1 \cup T_2[v_1, ..., v_{i'}]$ (qui est inférieur ou égal à $occ_M(c_{i'})$ car $col(V_T) \subseteq M$). Alors, $\pi_{T_1}\pi_{T_2}$ est un monôme multilinéaire de $P_{i_1,u}P_{i_2,v}$ et donc de $P_{i,u}$.

Inversement, soit $\pi_T = x_{v_1} y_{c_1,j_1} ... x_{v_i} y_{c_i,j_i}$, un monôme multilinéaire de $P_{i,u}$. Par définition de $P_{i,u}$, il existe un $1 \leq i' < i$ et un $v \in N(u)$ tel que π_T est un monôme multilinéaire de $P_{i',u}P_{i-i',v}$. Alors, il existe un monôme multilinéaire π_{T_1} dans $P_{i',u}$ et un monôme multilinéaire π_{T_2} dans $P_{i-i',v}$. Par l'hypothèse d'induction, il existe deux arbres $T_1 = (V_{T_1}, E_{T_1})$ et $T_2 = (V_{T_2}, E_{T_2})$ tels que (i) $u \in V_{T_1}$ et $v \in V_{T_2}$, (ii) $V_{T_1} = \{v_1, ..., v_{i'}\}$ et $V_{T_2} = \{v_1, ..., v_{i-i'}\}$ et (iii) $col(V_{T_1}) \subseteq M$ et $col(V_{T_2}) \subseteq M$. Comme π_T est multilinéaire, $col(V_{T_1} \cup V_{T_2}) \subseteq M$ et les sommets de V_{T_1} et V_{T_2} sont disjoints (si un même sommet u est présent dans V_{T_1} et V_{T_2}, alors x_u serait présent au moins deux fois et π_T ne serait pas multilinéaire). Par conséquent, $|V_{T_1} \cup V_{T_2}| = i' + (i - i') = i$ et $T = (V_{T_1} \cup V_{T_2}, E_{T_1} \cup E_{T_2} \cup \{u, v\})$ est une solution.

□

Peut-on obtenir un algorithme de complexité en temps $\mathcal{O}^*(2^k)$ pour EXACT GRAPH MOTIF quand le motif est un multi-ensemble, comme c'est déjà le cas quand le motif est colorful ? Il est possible que cette question puisse être résolue avec une extension des techniques de détection de monômes multilinéaires de Koutis et Williams.

4.4 Recherche de noyaux

On propose dans cette section quelques règles de réductions effectuées en temps polynomial (voir Section 1.3.3) pour le problème EXACT GRAPH MOTIF quand le motif est colorful. On rappelle que l'instance consiste en un graphe $G = (V, E)$ coloré sur

les sommets par une fonction $col : V \to C$, où C est un ensemble de k couleurs et qu'une solution est un sous-arbre $T = (V_T, E_T)$ tel que $|V_T| = k$ et $\forall u, v \in V_T$ distincts, $col(u) \neq col(v)$. On note $occ_G(c)$ le nombre d'occurrences de la couleur c dans G.

- $\forall v \in V$, si $d(v) = 0, V = V \setminus \{v\}$ et laisser k inchangé. Si un sommet v de G est isolé, il ne peut être dans aucune solution connexe.
- $\forall \{u, v\} \in E$ tel que $col(u) = col(v), E = E \setminus \{u, v\}$ et laisser k inchangé. Aucune solution colorful ne peut utiliser l'arête $\{u, v\}$ si u et v ont la même couleur. On note que le graphe obtenu est k-parti.
- Soit $dist(u, v)$ la longueur du plus court chemin entre u et v. $\forall v \in V, \forall c \in C$, si $\nexists u$ tel que $col(u) = c$ et $dist(u, v) < k$, alors $V = V \setminus \{v\}$ et laisser k inchangé. Si depuis un sommet v de G on ne peut pas atteindre toutes les couleurs en moins de k arêtes, alors v ne peut pas être dans une solution colorful de taille k.
- $\forall u, v \in V$ tel que $col(u) = col(v)$, si $N(u) \subseteq N(v), V = V \setminus \{u\}$ et laisser k inchangé. Si deux sommets u et v ont la même couleur et que le voisinage de u est compris dans celui de v, alors n'importe quelle solution colorful contenant u peut contenir v à la place.
- $\forall u \in V$, si $d(u) = 1$ et $occ_G(col(u)) = 1, V = V \setminus \{u\}, k = k - 1$. De plus, $\forall v'$ tel que $col(v') = col(v)$, où $\{u, v\} \in E$ et $v' \neq v, V = V \setminus \{v'\}$. Si un sommet u est la seule occurrence de sa couleur dans G, toute solution colorful doit contenir u. De plus, comme u est de degré 1, toute solution (connexe) doit contenir le voisin v de u et par conséquent, aucune solution colorful ne peut contenir un sommet de même couleur que v autre que v lui-même.
- Soit $\forall c \in C, V_c = \{v : col(v) = c\}$. Alors, $\forall c \in C$, il existe l_c composantes connexes dans $G[V \setminus V_c]$, notées $V_c^1, V_c^2, \ldots, V_c^{l_c}$. S'il existe un $1 \leq i \leq l_c$, tel que $\forall u \in V_c, |col(V \setminus V_c \cup \{u\})| < k$, alors $V = V \setminus V_c^i$. En effet, comme la solution est colorful, si depuis une composante connexe on ne peut pas ajouter une occurrence de c pour atteindre toutes les couleurs, aucun sommet de cette composante connexe ne peut être dans une solution.

Ces règles de réductions sont intéressantes pour réduire la taille des instances, afin d'accélérer le temps d'exécution d'un algorithme. Toutefois, nous n'avons pas obtenu de borne concernant la taille de l'instance réduite. Après l'obtention de ces règles, nous avons eu connaissance des travaux parus dans [BDFH09], supprimant tout espoir d'un noyau polynomial pour le problème EXACT GRAPH MOTIF. En effet, on peut effectuer une construction identique à celle effectuée pour le problème $k-$PATH (qui est présentée Section 1.3.3). Étant donnés t instances du problème EXACT GRAPH MOTIF, il existe une solution (connexe) dans l'union disjointe de ces t graphes si et seulement il existe une solution dans une des t instances.

Plus tard, il a été prouvé qu'un noyau de taille polynomial était improbable même sur des instances très contraintes du problème, quand G est un *comb-graph* et le motif colorful [ABRH+10]. Un *comb-graph* est un arbre où tous les sommets ont un degré

inférieur ou égal à trois et où les sommets de degré trois sont positionnés sur un même chemin (voir Figure 4.6). Sur cette classe particulière de graphe, la construction par union disjointe n'est plus valide car l'union de plusieurs *comb-graphs* ne donne pas un *comb-graph* (car non connexe). Pour lier deux *comb-graphs*, ils ajoutent un chemin de deux sommets, où ces deux sommets portent la couleur du sommet du *comb-graph* qui leur est adjacent. Ainsi, on obtient un *comb-graph*, où une solution colorful ne peut pas contenir des sommets d'instances différentes (voir Figure 4.6).

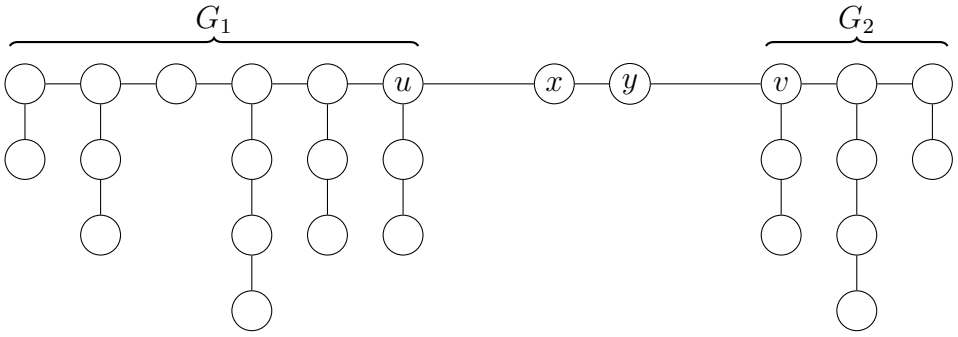


FIGURE 4.6 – Concaténation de deux *comb-graphs* G_1 et G_2 à l'aide d'un chemin de deux sommets, donnant un *comb-graph*. C'est un arbre où tous les sommets ont un degré inférieur ou égal à trois et où tous les sommets de degré trois sont sur un même chemin. En plaçant $col(x) = col(u)$ et $col(y) = col(v)$, une solution colorful ne peut pas contenir un sommet de G_1 et un sommet de G_2.

Une question ouverte concerne l'existence d'un noyau de taille $2^{o(k)}$ pour le problème EXACT GRAPH MOTIF.

4.5 Compter les motifs

Cette section, dont les résultats ont été publiés dans [GS10] (la version longue est acceptée pour publication [GS11]), est consacrée à l'étude de la version de comptage du problème vu précédemment.

#EXACT GRAPH MOTIF

• **Entrée** : Un graphe $G = (V, E)$, un ensemble de couleurs C, une fonction $col : V \to C$, un multi-ensemble M de taille k sur C.

• **Sortie** : Le nombre de sous-ensembles $V_T \subseteq V$ de G tels que (i) $|V_T| = k$, (ii) $G[V_T]$ soit connexe et (iii) $col(V_T) = M$.

Ce nouveau problème permet d'évaluer si un motif M est sur-représenté, ou sous-représenté, dans le réseau en comparant le nombre d'apparitions du motif par rapport au nombre attendu dans un réseau aléatoire (construit aléatoirement selon un modèle de probabilité), ce qui apporte des informations biologiques supplémentaires [SLS09].

Avant cette thèse, aucun résultat concernant la complexité paramétrée de ces problèmes n'était connu. Dans [ADH+08, GS09], les auteurs proposent des algorithmes approchés, pour compter le nombre d'occurrences d'un sous-graphe ayant une certaine topologie.

D'un point de vue biologique, étant donnée l'imprécision des données, il semble intéressant de pouvoir compter des motifs qui ne sont pas tous exactement identiques. Quelles contraintes donner sur chaque occurrence ? Ceci est, de plus, une question algorithmique intéressante.

Dans cette section, on montre que le problème #EXACT GRAPH MOTIF est de complexité paramétrée seulement si le motif est colorful. Cependant, lorsque le motif est un multi-ensemble, on prouve que le problème #EXACT GRAPH MOTIF est #W[1]-Difficile même lorsque seulement deux couleurs sont utilisées (on oriente le lecteur vers [FG04, FG06] pour la définition des classes de complexités relatives au comptage).

Pour montrer que le problème #EXACT GRAPH MOTIF est dans la classe FPT si le motif est colorful (Proposition 4.5.2), nous nous reposons sur un résultat plus général montrant que le problème #EXACT MULTILINEAR DETECTION est dans la classe FPT (Proposition 4.5.1). On précise que le problème MULTILINEAR DETECTION est dit exact si $|X| = k$.

#EXACT MULTILINEAR DETECTION

• **Entrée** : Un circuit arithmétique A représentant un polynôme P_A sur un ensemble X de k variables

• **Sortie** : Le nombre de monômes multilinéaires de degré k.

Étant donnés un ensemble X d'éléments et une fonction $f : 2^X \to \mathbb{Z}$, la *transformée de Möbius* [BHKK07] f' de f est, pour chaque $S \subseteq X$, la fonction

$$f'(S) = \sum_{T \subseteq S} f(T).$$

Étant donnée la transformée de Möbius f', la fonction f peut être retrouvée, pour chaque $S \subseteq X$, *via* la formule de l'*inversion de Möbius* [BHKK07]

$$f(S) = \sum_{T \subseteq S} (-1)^{|S \setminus T|} f'(T).$$

On considère maintenant que les circuits arithmétiques utilisés sont k-*bornés*. Un circuit A est k-*borné* si et seulement si P_A, le polynôme qu'il représente, possède seulement des monômes de degré inférieur ou égal à k. On montre dans [GS11] que l'on peut transformer n'importe quel circuit en un circuit k-borné, au prix d'un facteur $(k+1)^2$ sur la taille du nouveau circuit.

Le résultat suivant montre que l'on peut compter les solutions pour les circuits arithmétiques k-bornés avec k variables (et par conséquent pour les circuits généraux en ajoutant un facteur $\mathcal{O}(k^2)$ dans la complexité en temps). Remarquons que cet algorithme n'est pas randomisé, contrairement à celui proposé par le Théorème 1.3.1.

Proposition 4.5.1. *On peut résoudre le problème* #EXACT MULTILINEAR DETECTION *pour les circuits k-bornés par un algorithme de complexité en temps $\mathcal{O}(2^k|A|)$ et de complexité en espace $\mathcal{O}(|A|)$.*

Démonstration. Soit A le circuit arithmétique donné en entrée sur un ensemble X de k variables. Pour un monôme m, on note $Var(m)$ l'ensemble de ses variables. Pour $S \subseteq X$, on note N_S (resp. N'_S), le nombre de monômes m de P_A tel que $Var(m) = S$ (resp. $Var(m) \subseteq S$). On observe que pour tout $S \subseteq X$, on a $N'_S = \sum_{T \subseteq S} N_T$ (N' est donc la transformée de Möbius de N). Par conséquent, par l'inversion de Möbius, pour chaque $S \subseteq X$, $N_S = \sum_{T \subseteq S} (-1)^{|S \setminus T|} N'_T$.

Comme A est un circuit k-borné, N_X donne le nombre de monômes multilinéaires de P_A de degré k. Chaque valeur N'_S peut être calculée en évaluant le circuit A avec la fonction $\phi : X \to Z$ définit comme suit : $\phi(v) = 1$ si $v \in S$, $\phi(v) = 0$ si $v \notin S$. Cette fonction donne le bon nombre de monômes m tels que $Var(m) \subseteq S$. En effet, si toutes les variables d'un monôme m sont dans S, m est évalué à 1. En revanche, si une variable de m n'est pas dans S, m est évalué à 0. Par conséquent, N'_S peut être calculé en temps $\mathcal{O}(|A|)$. Par la fonction de l'inversion de Möbius, comme $|X| = k$, on peut alors calculer la valeur désirée N_X en temps $\mathcal{O}(2^k|A|)$ et en espace $\mathcal{O}(|A|)$. □

On mentionne que la Proposition 4.5.1 utilise le principe d'inclusion-exclusion [Kar82]. La Proposition 4.5.1 généralise plusieurs algorithmes de comptage basés sur cette notion, comme celui pour le problème #HAMILTONIAN PATH et ceux présents dans [Ned09]. En effet, les problèmes considérés dans ces articles peuvent être réduits au comptage de monômes multilinéaires de degré n pour des circuits à n variables (où n est habituellement le nombre de sommets dans le graphe), nous permettant d'obtenir des algorithmes ayant une complexité de $\mathcal{O}^*(2^n)$ en temps et une complexité en espace polynomiale.

On applique maintenant la Proposition 4.5.1 au problème #EXACT GRAPH MOTIF quand le motif est colorful. Rappelons que nous avons déjà défini dans la Propo-

sition 4.3.1 un circuit A pour le problème EXACT GRAPH MOTIF quand le motif est colorful ; nous avons à le modifier légèrement dans le cadre du comptage des solutions.

Proposition 4.5.2. *Le problème* #EXACT GRAPH MOTIF *peut se résoudre à l'aide d'un algorithme ayant une complexité en temps de* $\mathcal{O}(2^k k^3|E|)$ *et de complexité en espace* $\mathcal{O}(k^3|E|)$ *si le motif est colorful.*

Démonstration. Une *solution enracinée* pour une instance $\mathcal{I}$ du problème EXACT GRAPH MOTIF est une paire (u, T), où T est un arbre qui est une solution pour $\mathcal{I}$ et u est un sommet de T (pouvant être vu comme la racine de l'arbre). Les solutions de EXACT GRAPH MOTIF pour $\mathcal{I}$ sont aussi appelées *solutions non-enracinées*. Soit $N_r(\mathcal{I})$ et $N_u(\mathcal{I})$ le nombre de solutions enracinées, resp. non-enracinées pour $\mathcal{I}$. Nous allons montrer comment calculer $N_r(\mathcal{I})$ dans les bornes de complexité données dans l'énoncé de la proposition – le résultat reste valide car $N_u(\mathcal{I}) = \frac{N_r(\mathcal{I})}{k}$.

Pour calculer N_r, on observe d'abord qu'on ne peut pas appliquer la Proposition 4.5.1 au circuit A de la Proposition 4.3.1. En effet, ce circuit A compte les sous-arbres ordonnés et non les non-ordonnés. Par conséquent, nous devons modifier le circuit de la manière suivante : pour chaque sommet v de V_T, on examine ses fils par couleurs croissantes. Ceci nous mène à la définition du circuit arithmétique A' suivant : supposons sans perte de généralité que $C = \{1, ..., k\}$, on introduit les sommets $P_{i,j,u}$ pour chaque $1 \leq i \leq k, 1 \leq j \leq k+1, u \in V$, les variables x_i pour chaque $1 \leq i \leq k$ et on définit :

$$
\begin{aligned}
&P_{1,j,u} = x_{col(u)},\\
&P_{i,j,u} = 0 \text{ si } i \geq 2, j = k+1,\\
&P_{i,j,u} = P_{i,j+1,u} + \sum_{i'=1}^{i-1} \sum_{v \in N(u) \text{ t.q. } col(v)=j} P_{i',j+1,u} P_{i-i',1,v} \text{ si } i \geq 2, 1 \leq j \leq k.
\end{aligned}
$$

On introduit aussi $P = \sum_{u \in V} P_{k,1,u}$ comme étant la racine du circuit A. Étant donnés $1 \leq i, j \leq k$ et $u \in V$, on note $\mathcal{S}_{i,j,u}$ l'ensemble des paires (u, T) où (i) T est un sous-arbre coloré distinctement pour $\mathcal{I}$, contenant u et ayant i sommets, (ii) les voisins de u dans T ont des couleurs supérieures ou égales à j. On peut montrer par induction sur i qu'il y a une bijection entre $\mathcal{S}_{i,j,u}$ et les monômes multilinéaires de $P_{i,j,u}$. Par conséquent, le nombre de monômes multilinéaires de P est égal à N_r ; comme $|A'| = \mathcal{O}(k^3|E|)$ et comme A' est k-borné, par la Proposition 4.5.1, on calcule N_r avec une complexité en temps de $\mathcal{O}(2^k k^3|E|)$ et une complexité en espace de $\mathcal{O}(k^3|E|)$. □

La proposition précédente n'est valide que si le motif est colorful. De manière assez surprenante, lorsque le motif est un multi-ensemble, on ne peut pas résoudre ce problème avec un algorithme de complexité paramétrée, contrairement au problème de décision. La preuve, assez technique, se trouve dans [GS10, GS11]. Elle consiste en la réduction du problème #W[1]-Difficile #MULTICOLORED CLIQUE vers le problème

#EXACT GRAPH MOTIF, *via* deux problèmes intermédiaires introduits spécialement dans le cadre de cette réduction [GS10].

Proposition 4.5.3. *Le problème* #EXACT GRAPH MOTIF *est* #W[1]*-Difficile pour le paramètre* k*, même lorsque* M *n'utilise que deux couleurs différentes.*

Le problème #EXACT GRAPH MOTIF est #W[1]-Difficile pour les motifs contenant deux couleurs. La complexité du problème avec une seule couleur dans le motif reste ouverte. On remarque que ce problème est équivalent à compter les sous-arbres contenant k sommets dans un graphe (non-coloré).

Chapitre 5

Vers une définition plus souple du problème GRAPH MOTIF

Contenu

Nous avons déjà vu que les réseaux biologiques sont construits depuis des données expérimentales et que celles-ci souffrent souvent d'imprécision. De plus, il existe des variations dues à l'évolution. Par conséquent, rechercher des occurrences exactes, comme dans les problèmes vus dans le chapitre précédent, implique un fort risque de ne pas trouver de solution. Dans ce chapitre, nous voyons des variantes du problème, autorisant d'avantage de flexibilité.

Plus précisément, nous utilisons dans un premier temps la technique de la détection de monômes multilinéaires afin d'obtenir des algorithmes de complexité paramétrée pour quatre variantes du problème EXACT GRAPH MOTIF. Ceci nous permet d'obtenir une complexité en temps plus faible que celle des algorithmes existants et une complexité en espace polynomiale (lorsque les algorithmes de la littérature utilisent un espace exponentiel).

Dans un deuxième temps, nous nous intéressons à l'approximation de deux variantes. Malheureusement, nos résultats sont négatifs et montrent qu'il est impossible d'obtenir un algorithme d'approximation ayant un ratio inférieur à une certaine borne, même pour des instances très contraintes.

Enfin, nous proposons une évolution au problème GRAPH MOTIF étudié précédemment, en modifiant la contrainte de simple connexité.

5.1 Résultats de complexité paramétrée

5.1.1 Quand des insertions et des délétions sont autorisées

Une première manière de rendre ces problèmes plus souples consiste à ne pas rechercher une solution contenant toutes les couleurs du motif, mais d'autoriser qu'un certain nombre d'entre elles soit absent. Ces couleurs sont supprimées de la solution par rapport au motif (Bruckner *et al.* parlent de *délétions* [BHK+09]). Un exemple se trouve Figure 5.1. Le problème suivant autorise les délétions.

MAXIMUM GRAPH MOTIF

• **Entrée** : Un graphe $G = (V, E)$, un entier k, un ensemble de couleurs C, une fonction $col : V \to C$, un multi-ensemble M sur C.

• **Sortie** : Un sous-ensemble $V_T \subseteq V$ tel que (i) $|V_T| = k$, (ii) $G[V_T]$ soit connexe et (iii) $col(V_T) \subseteq M$.

Par rapport au problème EXACT GRAPH MOTIF, la taille du multi-ensemble M n'est pas fixée à k. Par conséquent, on demande seulement que l'ensemble des couleurs de la solution soit compris dans M.

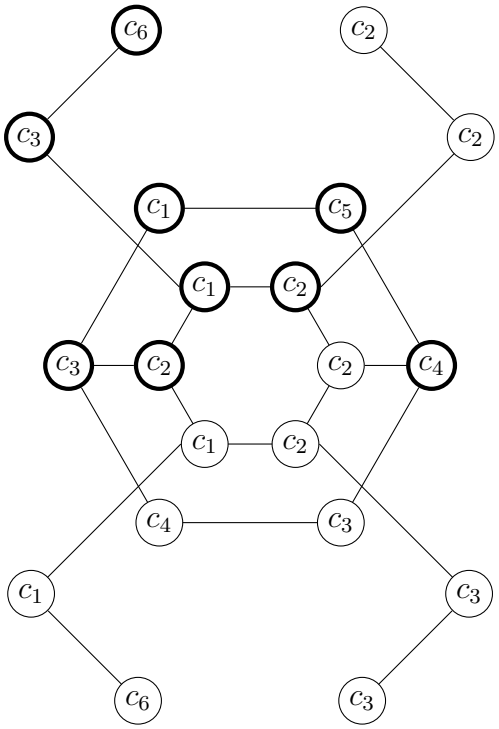


FIGURE 5.1 – Exemple fictif d'un graphe G coloré sur les sommets par l'ensemble de couleurs $C = \{c_i : 1 \leq i \leq 6\}$. On trouve en gras dans le réseau une solution possible $col(V_T) = \{c_1, c_1, c_2, c_2, c_3, c_3, c_4, c_5, c_6\}$ pour le problème MAXIMUM GRAPH MOTIF avec $k = 9$ et le motif $M = \{c_1, c_1, c_2, c_2, c_3, c_3, c_4, c_5, c_6, c_6\}$ (deux occurrences de la couleur c_6 sont demandées dans M, mais l'une d'entre elles n'est pas dans la solution).

Puisque l'on chercher à minimiser le nombre de délétions (ou maximiser la taille de la solution), les problèmes correspondants sont d'optimisation – on rejoint d'ailleurs la définition donnée par Dondi *et al.* du problème d'optimisation MAXIMUM MOTIF [DFV09]. Cependant, nous avons choisi de décrire les problèmes correspondants sous leur forme de décision, où le nombre de délétions est exactement égal à $|M| - k$.

Dondi *et al.* [DFV09] présentent un algorithme pour résoudre MAXIMUM GRAPH MOTIF utilisant la technique du color-coding avec une complexité en temps implicite de $\mathcal{O}((32e^2)^k km)$ et une complexité exponentielle en espace. Lorsque le motif est colorful, Bruckner *et al.* fournissent un algorithme ayant une complexité en temps de $\mathcal{O}(3^k|E|)$ [BHK+09]. Ils observent en effet que les délétions peuvent être directement gérées par leur algorithme pour le problème EXACT GRAPH MOTIF quand le motif est colorful, en vérifiant dans la table de programmation dynamique la valeur de chaque case correspondant à $V'_T \subseteq V_T$, tel que $|V'_T| \geq |M| - N_{del}$, où N_{del} est un nombre prédéfini de délétions autorisées.

Observons que les algorithmes déjà proposés pour le problème EXACT GRAPH MOTIF sont toujours valides pour résoudre le problème MAXIMUM GRAPH MOTIF.

Proposition 5.1.1. *On peut résoudre le problème* MAXIMUM GRAPH MOTIF *par un algorithme randomisé avec une complexité en espace de $\tilde{\mathcal{O}}(k^2|E|)$, et une complexité en temps de $\tilde{\mathcal{O}}(2^k k^2|E|)$ si le motif est colorful, ou $\tilde{\mathcal{O}}(4^k k^2|E|)$ si le motif est un multi-ensemble.*

Démonstration. Quelque soit le nombre de variables dans le polynôme, la complexité du problème MULTILINEAR DETECTION reste identique. Par conséquent, les preuves des Propositions 4.3.1 et 4.3.2 restent valides pour le problème MAXIMUM GRAPH MOTIF. □

Peut-on envisager une variante du problème MAXIMUM GRAPH MOTIF où l'on chercherait à minimiser les différences (au carré) entre l'occurrence d'une couleur dans le motif et l'occurrence de cette couleur dans la solution ? Ceci permettrait d'être potentiellement plus « proche » du motif demandé. Par exemple, si le motif est composé de dix rouges, dix verts et dix bleus, on privilégie une solution apportant huit rouges, huit verts et huit bleus plutôt que dix rouges, dix verts et quatre bleus.

Dans l'article introduisant le problème EXACT GRAPH MOTIF, Lacroix *et al.* [LFS06] précisent que la notion de *gap* est nécessaire (ces gaps sont nommés *insertions* dans [BHK+09]). Ces sommets insérés ne correspondent pas à des éléments du motif, mais sont nécessaires pour satisfaire la contrainte de connexité (voir Figure 5.2). Bruckner *et al.* indiquent que les insertions sont préférables aux délétions car elles correspondent mieux à des données incomplètes [BHK+09].

GRAPH MOTIF WITH GAPS

• **Entrée** : Un graphe $G = (V, E)$, un entier k, un entier N_{ins} bornant le nombre d'insertions, un ensemble de couleurs C, une fonction $col : V \to C$ et un multi-ensemble M sur C.

• **Sortie** : Un sous-ensemble $V_T \subseteq V$ tel que (i) $|V_T| \leq k + N_{ins}$, (ii) $G[V_T]$ soit connexe et (iii) il existe un ensemble $S \subseteq V_T$ de taille k tel que $col(S) \subseteq M$.

Encore une fois, nous avons choisi de définir ce problème d'optimisation dans sa version de décision. Ainsi, une solution à GRAPH MOTIF WITH GAPS contient exactement $|V_T| - k$ insertions (dont la couleur n'est pas prise en compte) et exactement $|M| - k$ délétions.

Lorsque le motif est colorful, Bruckner *et al.* proposent un algorithme ayant une complexité en temps de $\mathcal{O}(3^k N_{ins}|E|)$, où N_{ins} est un nombre prédéfini d'insertions autorisées et une complexité en espace exponentielle [BHK+09]. On propose ci-après un algorithme utilisant le problème MULTILINEAR DETECTION pour résoudre le problème GRAPH MOTIF WITH GAPS, avec une complexité en espace polynomiale. Ce résultat, publié dans [GS10], est donc la première approche pour résoudre le problème GRAPH MOTIF WITH GAPS, lorsque le motif est un multi-ensemble.

Proposition 5.1.2. *On peut résoudre le problème* GRAPH MOTIF WITH GAPS *par un algorithme randomisé ayant une complexité en temps de* $\tilde{\mathcal{O}}(4^k(k + N_{ins})^2|E|)$ *et une complexité en espace de* $\tilde{\mathcal{O}}((k + N_{ins})^2|E|)$.

Démonstration. Pour chaque couleur $c \in C$ où $occ_M(c) = m$, on introduit les variables $y_{c,1}, ..., y_{c,m}$, ainsi qu'un sommet du circuit arithmétique $Q_c = y_{c,1} + ... + y_{c,m}$. Pour chaque sommet $u \in V$, on introduit une variable x_u. Le circuit arithmétique A est construit comme suit :

$$P_{1,u} = 1 + x_u Q_{col(u)},$$
$$P_{i,u} = \sum_{i'=1}^{i-1} \sum_{v \in N(u)} P_{i',u} P_{i-i',v} \text{ si } i > 1,$$

et $P = \sum_{u \in V} \sum_{i=1}^{k+N_{ins}} P_{i,u}$ est la racine de A.

On note que seuls $P_{1,u}$ et P sont modifiés par rapport à la construction du circuit arithmétique de la Proposition 4.3.2. De manière informelle, l'ajout de la constante 1 à chaque $P_{1,u}$ permet d'ignorer certains sommets du sous-arbre, en autorisant seulement la sélection d'un ensemble S de k sommets tel que $col(S) \subseteq M$.

En appliquant le Théorème 1.3.1 sur l'instance $(A, 2k)$ et en observant que $|A| = \mathcal{O}((k + N_{ins})^2|E|)$, on donne une solution à $(A, 2k)$ en temps $\tilde{\mathcal{O}}(4^k(k + N_{ins})^2|E|)$, en utilisant $\tilde{\mathcal{O}}((k + N_{ins})^2|E|)$ d'espace.

On prouve que cette construction est correcte par une induction similaire à celle de la Proposition 4.3.2. Lorsque l'on considère deux arbres T_1 et T_2 obtenus depuis $P_{i',u}$ et $P_{i-i',v}$, leurs sommets sélectionnés sont bien distincts, mais peuvent avoir des sommets insérés en commun. On peut alors trouver un sous-ensemble de $E(T_1) \cup E(T_2) \cup \{u, v\}$ formant un arbre contenant tous les sommets sélectionnés de T_1 et T_2. □

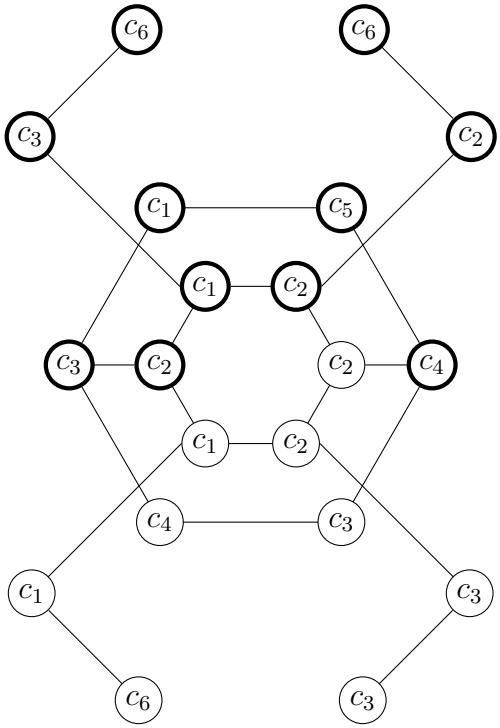


FIGURE 5.2 – Exemple fictif d'un graphe G coloré sur les sommets par l'ensemble de couleurs $C = \{c_i : 1 \leq i \leq 6\}$. On trouve en gras dans le réseau une solution possible $col(V_T) = \{c_1, c_1, c_2, c_2, c_2, c_3, c_3, c_4, c_5, c_6, c_6\}$ pour le problème GRAPH MOTIF WITH GAPS avec $k = 10$, $N_{ins} = 1$ et le motif $M = \{c_1, c_1, c_2, c_2, c_3, c_3, c_4, c_5, c_6, c_6, c_6\}$. Un sommet de couleur c_2 est inséré en plus dans la solution par rapport au motif, tandis qu'une des trois occurrences demandées de la couleur c_6 n'est pas dans la solution.

On observe qu'on ne peut pas espérer trouver un algorithme appartenant à la classe FPT avec pour paramètre le nombre N_{ins} d'insertions maximum ou le nombre N_{del} de délétions maximum . En effet, la complexité d'un tel algorithme serait alors de $f(N_{ins})|V|^c$, avec c une constante – ceci contredisant la difficulté du problème EXACT GRAPH MOTIF où $N_{ins} = 0$.

5.1.2 Quand un ensemble de couleurs est associé aux sommets du graphe

Dans la version originale du problème introduit par Lacroix *et al.* [LFS06], plusieurs couleurs peuvent être associées à chaque sommet du réseau. Il est en effet pertinent biologiquement de considérer plusieurs fonctions pour une même réaction dans un réseau métabolique ou de considérer qu'une protéine puisse être homologue avec plusieurs autres protéines dans un réseau d'interaction entre protéines. Ce problème est formalisé dans [BFKN08].

LIST-COLORED GRAPH MOTIF

• **Entrée** : Un graphe $G = (V, E)$, un entier k, un ensemble de couleurs C, un multi-ensemble M sur C, une fonction $col : V \to 2^C$ donnant pour chaque sommet de V l'ensemble des couleurs associé.

• **Sortie** : Un sous-ensemble $V_T \subseteq V$ tel que (i) $|V_T| = k$, (ii) $G[V_T]$ soit connexe et (iii) une bijection $f : V_T \to M$ telle que $\forall v \in V_T, f(v) \in col(v)$.

Betzler *et al.* [BFKN08] proposent un algorithme randomisé utilisant la technique du color-coding (d'une manière différente par rapport à l'algorithme présenté Section 4.3.2.1) afin d'obtenir une complexité en temps de $\mathcal{O}(10.88^k k^2|E|)$ et utilisant un espace exponentiel.

L'utilisation de la détection de monômes multilinéaires permet d'obtenir un algorithme de même complexité que lorsqu'une seule couleur est attribuée à chaque sommet du graphe. La complexité en temps est donc beaucoup plus faible que les algorithmes déjà connus. De plus, la complexité en espace est polynomiale. Ce résultat se trouve dans [GS11].

Proposition 5.1.3. *On peut résoudre le problème* LIST-COLORED GRAPH MOTIF *par un algorithme randomisé ayant une complexité en temps de* $\tilde{\mathcal{O}}(4^k k^2|E|)$ *et une complexité en espace de* $\tilde{\mathcal{O}}(k^2|E|)$.

Démonstration. Pour chaque couleur $c \in C$ où $occ_M(c) = m$, on introduit les variables $y_{c,1}, ..., y_{c,m}$, ainsi qu'un sommet du circuit arithmétique $Q_c = y_{c,1} + ... + y_{c,m}$. Pour chaque sommet $u \in V$, on introduit une variable x_u. Le circuit arithmétique A est construit comme suit :

$$P_{1,u} = x_u \sum_{c \in col(u)} Q_u,$$

$$P_{i,u} = \sum_{i'=1}^{i-1} \sum_{v \in N(u)} P_{i',u} P_{i-i',v}, \text{ si } i > 1,$$

et $P = \sum_{u \in V} P_{k,u}$ comme racine de A. On note que seul le sommet $P_{1,u}$ est modifié par rapport à la construction de la Proposition 4.3.2. Intuitivement, le sommet u représenté par la variable x_u peut être associé à n'importe quelle couleur de $col(u)$. Puisque l'on cherche un monôme multilinéaire, la variable x_u ne peut apparaître qu'une seule fois dans la solution et ainsi nous avons bien une bijection entre les couleurs de la solution et le motif.

Comme $|A| = \mathcal{O}(k^2|E|)$, en utilisant le Théorème 1.3.1, on obtient un algorithme randomisé pour le problème LIST-COLORED GRAPH MOTIF ayant une complexité en temps de $\tilde{\mathcal{O}}(4^k k^2|E|)$ et une complexité en espace de $\tilde{\mathcal{O}}(k^2|E|)$. □

5.1.3 Quand un poids est associé aux arêtes du graphe

Pour répondre à des problématiques différentes, une version où les arêtes du graphe ont un poids a été introduite par Böcker *et al.* [BRS09]. La *spectrométrie de masse* est une technique largement utilisée pour l'identification de molécules. Pour leur identification, elles sont fragmentées, puis la masse et l'abondance des ions sont relevées pour chaque fragment. Déterminer la structure de ces fragments manuellement prend du temps. Par conséquent, les auteurs de [BRS09] cherchent à automatiser cette tâche, en cherchant, dans un graphe correspondant à une structure moléculaire, une sous-structure correspondant au motif, qui est ici une certaine formule chimique. De plus, le poids (correspondant à des énergies) sur les arêtes de cette structure doit être minimal. Ceci mène à la définition du problème suivant (sous sa version de décision) :

WEIGHTED GRAPH MOTIF

• **Entrée** : Un graphe $G = (V, E)$, deux entiers k et r, un ensemble de couleurs C, un multi-ensemble M sur C, une fonction $col : V \to C$, une fonction de pondération $w : E \to \mathbb{N}$.

• **Sortie** : Un sous-graphe $T = (V_T, E_T)$ de G tel que (i) $|V_T| = k$, (ii) $G[V_T]$ soit connexe, (iii) $col(V_T) \subseteq M$ et (iv) $\sum_{e \in E_T} w(e) \leq r$.

On observe que dans leur définition du problème WEIGHTED GRAPH MOTIF, les auteurs de [BRS09] minimisent la somme des poids des arêtes $\{u, v\}$ où seulement u appartient à la solution. Pour résoudre WEIGHTED GRAPH MOTIF, Böcker *et al.* [BRS09] proposent une heuristique basée sur la décomposition arborescente, un algorithme de *branch and bound*, dont la complexité en temps $\mathcal{O}^*(|V|^b)$ est exponentielle selon le nombre de bonds b (précisé petit selon les auteurs), ainsi qu'un algorithme de complexité paramétrée, dont la complexité en temps est de $\mathcal{O}^*(2^{k+k\Delta})$, où k est la taille du motif et Δ est le degré du graphe G, prétendu borné dans le type de structure utilisé.

On observe que le problème WEIGHTED GRAPH MOTIF contient comme cas particulier le problème MINIMUM CC GRAPH MOTIF introduit par Dondi *et al.* sous le nom de MIN-CC [DFV07], que l'on décrit sous sa forme de décision.

MINIMUM CC GRAPH MOTIF

• **Entrée** : Un graphe $G = (V, E)$, un entier r, un ensemble de couleurs C, une fonction $col : V \to C$, un multi-ensemble M sur C.

• **Sortie** : Un sous-ensemble $V' \subseteq V$ tel que (i) $col(V') = M$ et (ii) $G[V']$ contienne au plus r composantes connexes.

En effet, on peut réduire en temps polynomial le problème MINIMUM CC GRAPH MOTIF au problème WEIGHTED GRAPH MOTIF. Étant donné le graphe G d'une instance du problème MINIMUM CC GRAPH MOTIF, on construit un graphe complet G' avec le même ensemble de sommets, où le poids des arêtes correspondant à G est nul et où l'on ajoute des arêtes de poids 1 pour rendre G' complet. On voit qu'il existe une solution pour G avec au plus r composantes connexes si et seulement si il existe une solution pour G' de poids au plus r.

Le problème MINIMUM CC GRAPH MOTIF offre ainsi une alternative au problème GRAPH MOTIF WITH GAPS. Il y a moins de restrictions car la distance entre deux sommets de la solution est potentiellement non-bornée.

Concernant la complexité paramétrée du problème MINIMUM CC GRAPH MOTIF, Dondi *et al.* [DFV07] prouvent que le problème est dans la classe FPT à l'aide de la technique du color-coding. Betzler *et al.* montrent qu'il est possible d'adapter leur algorithme pour le problème EXACT GRAPH MOTIF afin d'obtenir un algorithme pour le problème MINIMUM CC GRAPH MOTIF ayant une complexité en temps de $\mathcal{O}^*(4.32^k)$ et une complexité exponentielle en espace [BFKN08].

Il n'est pas possible d'obtenir un algorithme appartenant à la classe FPT avec pour paramètre le nombre de composantes connexes, même lorsque G est un arbre, sans contredire la NP-Difficulté du problème EXACT GRAPH MOTIF. Betzler *et al.* montrent de plus que même lorsque G est un chemin, le problème reste W[1]-Complet quand le paramètre est le nombre de composantes connexes [BFKN08].

Dondi *et al.* [DFV07] montrent que le problème MINIMUM CC GRAPH MOTIF est APX-Difficile si G est un chemin (même lorsque le motif est colorful et que chaque couleur apparaît au plus deux fois dans G). Ils montrent également qu'il n'existe pas d'algorithme d'approximation ayant un ratio inférieur à $c \log(|V|)$, avec c une constante, si G est un arbre (et le motif colorful).
Peut-on trouver un ratio d'approximation constant pour le problème MINIMUM CC GRAPH MOTIF si le motif est colorful et G est un chemin où chaque couleur apparaît au plus deux fois ? S'il est évident que ce n'est pas une question biologique de première importance, cela reste un challenge algorithmique intéressant.

En utilisant le problème MULTILINEAR DETECTION, on améliore significativement les complexités temps et espace pour le problème WEIGHTED GRAPH MOTIF, mais également pour le problème MINIMUM CC GRAPH MOTIF. Ce résultat est publié dans [GS10]. On considère maintenant le graphe G comme complet, afin d'avoir une pondération pour n'importe quelle paire de sommets $u, v \in V$.

Proposition 5.1.4. *On peut résoudre le problème* WEIGHTED GRAPH MOTIF *avec un algorithme randomisé ayant une complexité en espace de $\tilde{\mathcal{O}}(k^2r^2|E|)$ et une complexité en temps de $\tilde{\mathcal{O}}(2^k k^2 r^2|E|)$ si le motif est colorful ou $\tilde{\mathcal{O}}(4^k k^2 r^2|E|)$ si le motif est un multi-ensemble.*

Démonstration. Nous allons détailler la construction uniquement pour le cas où le motif est colorful, car les modifications nécessaires quand le motif est un multi-ensemble sont exactement identiques à celles effectuées pour le circuit de la Proposition 4.3.2.

La construction du circuit arithmétique A est similaire à celui de la Proposition 4.3.1. L'ensemble des variables est $\{x_c : c \in C\}$ (une variable pour chaque couleur) et nous introduisons les sommets $P_{i,j,u}$, pour $1 \leq i \leq k$ et $0 \leq j \leq r$, dont les monômes multilinéaires correspondent aux sous-arbres colorful ayant i sommets dont u et un poids total inférieur à j.

Les sommets du circuit sont les suivants :

$$P_{1,j,u} = x_{col(u)},$$
$$P_{i,j,u} = \sum_{i'=1}^{i-1} \sum_{v \in N(u)} \sum_{j'=0}^{j-w(\{u,v\})} P_{i',j',u} P_{i-i',j-j'-w(\{u,v\}),v} \text{ si } i > 1,$$

et $P = \sum_{u \in V} P_{k,r,u}$ comme racine de A.

L'instance pour le problème MULTILINEAR DETECTION est (A, k). Comme $|A| = \mathcal{O}(k^2r^2|E|)$, en utilisant le Théorème 1.3.1, on obtient un algorithme randomisé pour le problème WEIGHTED GRAPH MOTIF de complexité en temps $\tilde{\mathcal{O}}(2^k k^2 r^2|E|)$ et de complexité en espace $\tilde{\mathcal{O}}(k^2r^2|E|)$.

On prouve que cette construction est correcte en montrant qu'étant donnés $1 \leq i \leq k, 0 \leq j \leq r$ et $u \in V$, alors $x_{c_1}...x_{c_d}$ est un monôme multilinéaire de $P_{i,j,u}$ si et seulement si (i) $d = i$ et (ii) il existe $T = (V_T, E_T)$, un sous-arbre colorful de G avec $u \in V_T, col(V_T) = \{c_1, ..., c_d\}$ et $\sum_{e \in E_T} w(e) \leq j$.

□

5.2 Résultats d'approximation

Les problèmes vus dans la section précédente apportent plus de souplesse d'un point de vue biologique. Cependant, ils contiennent le problème EXACT GRAPH MOTIF en cas particulier. Par conséquent, les résultats de difficulté de ce dernier s'appliquent également à toutes les variantes vues dans la section précédente. Comme certains

sont des problèmes d'optimisation, une alternative à la complexité paramétrée est envisageable – on peut en effet tenter de donner des résultats approchés, avec un ratio d'erreur contrôlé. Malheureusement, nous avons, avec Romeo Rizzi, obtenu des résultats négatifs, non publiés, concernant l'approximation de ces problèmes, même lorsque les instances sont très contraintes.

5.2.1 Maximiser la taille de la solution

On propose d'étudier dans cette section la version naturelle d'optimisation du problème MAXIMUM GRAPH MOTIF, appelé MAXIMUM MOTIF par [DFV09], où l'on cherche à maximiser la taille de la solution.

Il est connu que le problème MAXIMUM GRAPH MOTIF est APX-Difficile même lorsque G est un arbre de degré maximum trois, le motif est colorful et chaque couleur n'apparaît que deux fois dans G [DFV09] (à titre de comparaison, on remarque que dans ces mêmes conditions, le problème EXACT GRAPH MOTIF est polynomial, comme le montre la Proposition 4.2.1). De plus, il est impossible de trouver un ratio d'approximation constant, sauf si $\mathsf{P} = \mathsf{NP}$, même lorsque G est un arbre et M est colorful [DFV09]. Dans le cas spécifique où G est un arbre, le motif colorful et chaque couleur n'apparaît que deux fois, on peut s'interroger sur la possibilité d'approximer à ratio constant le problème.

Malheureusement, il s'avère que le problème continue d'être difficile à approximer, même dans ce cas très spécifique. Plus précisément, on ne peut pas approximer le problème MAXIMUM GRAPH MOTIF à un ratio inférieur à $|V|^{\frac{1}{3}-\epsilon}, \forall \epsilon > 0$, même lorsque G est un arbre, le motif est colorful et si chaque couleur apparaît dans G au plus deux fois.

Nous nous intéressons spécifiquement au cas où le motif est colorful (notre résultat négatif s'étend directement au cas où le motif est un multi-ensemble).

Pour obtenir ce résultat, nous allons montrer que le problème MAXIMUM GRAPH MOTIF est aussi difficile à approximer que le problème MAXIMUM INDEPENDENT SET, dont on rappelle la définition.

MAXIMUM INDEPENDENT SET

- **Entrée** : Un graphe $G = (V, E)$.
- **Sortie** : Un sous-ensemble des sommets $V' \subseteq V$ tel que V' est un ensemble indépendant, c'est-à-dire qu'il n'existe pas deux sommets $u, v \in V'$ tels que $\{u, v\} \in E$.
- **Mesure** : La taille de l'ensemble indépendant, c'est-à-dire de V'.

Pour plus de lisibilité, nous notons que l'instance du problème MAXIMUM INDEPENDENT SET est un graphe $G_I = (V_I, E_I)$. La preuve se déroule en quatre étapes. Nous

décrivons la construction de l'instance $\mathcal{I}' = (G, C, col)$ pour le problème MAXIMUM GRAPH MOTIF depuis une instance $\mathcal{I} = (G_I)$ du problème MAXIMUM INDEPENDENT SET (le motif M n'est pas défini explicitement, il est néanmoins possible de considérer que $M = C$). Ensuite, nous prouvons que l'on peut construire en temps polynomial une solution pour $\mathcal{I}'$ depuis une solution pour $\mathcal{I}$ et, inversement, que l'on peut construire en temps polynomial une solution pour $\mathcal{I}$ depuis une solution pour $\mathcal{I}'$. Enfin, nous montrons que s'il existe un algorithme d'approximation de ratio r pour le problème MAXIMUM GRAPH MOTIF, alors il existe un algorithme d'approximation de ratio r pour le problème MAXIMUM INDEPENDENT SET.

Avant d'expliciter la construction, on considère un ordre total des arêtes de G, puis on se munit d'une fonction $\mathrm{adj} : V_I \to 2^{E_I}$, donnant pour un sommet $v \in V_I$ la liste ordonnée des arêtes où v est impliqué (donc de taille degré de v, $d(v)$). Étant donné cet ordre, on considère que $\mathrm{adj}(v)[i]$ donne la i^{eme} arête où v est impliqué.

Depuis le graphe $G_I = (V_I, E_I)$, on construit le graphe $G = (V, E)$ de la manière suivante (voir Figure 5.3) :

– $V = \{r\} \cup$

$\{v_i^e : 1 \leq i \leq |V_I|, e \in \mathrm{adj}(v_i)\} \cup$

$\{v_i^j : 1 \leq i \leq |V_I|, 1 \leq j \leq |V_I|^2\},$

– $E = \{\{r, v_i^{\mathrm{adj}(v_i)[1]}\} : 1 \leq i \leq |V_I|\} \cup$

$\{\{v_i^{\mathrm{adj}(v_i)[j]}, v_i^{\mathrm{adj}(v_i)[j+1]}\} : 1 \leq i \leq |V_I|, 1 \leq j < d(v_i)\} \cup$

$\{\{v_i^{\mathrm{adj}(v_i)[d(v_i)]}, v_i^1\} : 1 \leq i \leq |V_I|\} \cup$

$\{\{v_i^j, v_i^{j+1}\} : 1 \leq i \leq |V_I|, 1 \leq j < |V_I|^2\}.$

Intuitivement, r est la racine de G. Il y a $|V_I|$ chemins connectés à cette racine. Chacun de ces chemins représente un sommet de G_I et est de longueur $d(v_i)+|V_I|^2$. On remarque que $|V| = 1 + 2|E_I| + |V_I|.|V_I|^2$ (il y a deux sommets impliqués dans chaque arête, donc $\sum_{v \in V_I} d(v) = 2|E_I|$).

Voyons maintenant la définition de l'ensemble de couleurs C et de la fonction de coloration $col : V \to C$. L'ensemble des couleurs est $C = \{c_r\} \cup \{c_e : e \in E_I\} \cup \{c_i^j : 1 \leq i \leq |V_I|, 1 \leq j \leq |V_I|^2\}$. En considérant $M = C$, le motif est clairement colorful. La coloration des sommets de G est la suivante : $col(r) = c_r$, $\forall e = (v_i, v_j) \in E_I, col(v_i^e) = col(v_j^e) = c_e$, $\forall 1 \leq i \leq |V_I|, 1 \leq j \leq |V_I|^2, col(v_i^j) = c_i^j$. De manière informelle, les sommets r et les sommets v_i^j ont tous des couleurs différentes. De plus, pour chaque arête $e = \{v_i, v_j\}$, une copie du sommet v_i et une copie du sommet v_j ont la même couleur c_e, celle de l'arête. On voit clairement que par construction, G est un arbre où chaque couleur apparaît au plus deux fois.

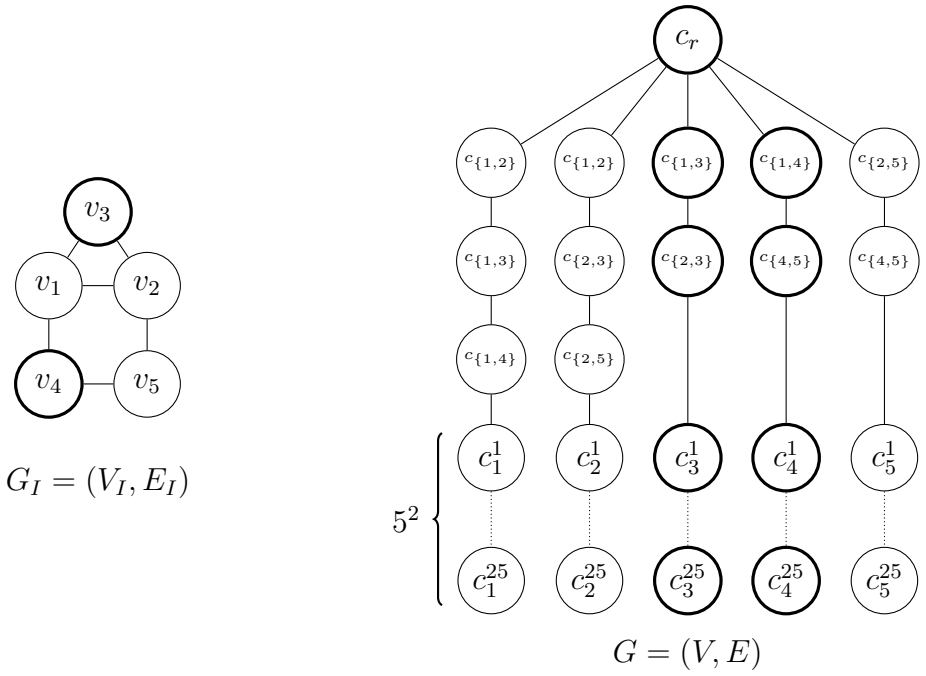


FIGURE 5.3 – Construction du graphe $G = (V, E)$ depuis l'instance du problème MAXIMUM INDEPENDENT SET $G_I = (V_I, E_I)$. Pour des raisons de lisibilité, nous avons seulement indiqué la couleur des sommets (pas leur nom) de G. Pour une solution pour G_I en gras, nous avons indiqué une solution pour MAXIMUM GRAPH MOTIF dans G en gras.

On montre maintenant comment obtenir une solution pour $\mathcal{I}'$ depuis une solution de $\mathcal{I}$.

Lemme 5.2.1. *S'il existe une solution $V_I' \subseteq V_I$ pour $\mathcal{I}$, alors il existe une solution $T = (V_T, E_T)$ pour $\mathcal{I}'$ telle que $|V_T| \geq |V_I'|.|V_I|^2$.*

Démonstration. On construit V_T de la manière suivante : $V_T = \{r\} \cup \{v_i^e, v_i^j : v_i \in V_I', e \in \text{adj}(v_i), 1 \leq j \leq |V_I|^2\}$. En d'autres termes, on ajoute dans V_T la racine ainsi que l'ensemble des chemins correspondant aux sommets de V_I' (voir aussi Figure 5.3).

Prouvons maintenant que V_T est bien une solution pour $\mathcal{I}'$ telle que $|V_T| \geq |V_I'|.|V_I|^2$. La racine implique que $G[V_T]$ est connexe. De plus, les couleurs de V_T sont toutes distinctes, la solution construite est colorful. En effet, s'il existe $u, v \in V_T$ tels que $col(u) = col(v)$, alors $\{u, v\} \in E$, ce qui contredit le fait que V_I' soit une solution pour le problème MAXIMUM INDEPENDENT SET. Enfin, on donne une borne sur la taille de V_T en observant que pour chaque $v \in V_I'$, on ajoute l'ensemble des sommets du chemin correspondant à v, de taille $d(v) + |V_I|^2 \geq |V_I|^2$. □

Montrons maintenant comment obtenir une solution pour $\mathcal{I}$ depuis une solution de $\mathcal{I}'$.

Lemme 5.2.2. *S'il existe une solution $T = (V_T, E_T)$ pour $\mathcal{I}'$, alors il existe une solution $V_I' \subseteq V_I$ pour $\mathcal{I}$ telle que $|V_I'| \geq \left\lceil \frac{|V_T| - 2|E_I| - 1}{|V_I|^2} \right\rceil$.*

Démonstration. Pour $1 \leq i \leq |V_I|$, on ajoute v_i dans V'_I si et seulement si tous les sommets $v_i^j, 1 \leq j \leq |V_I|^2$ et $v_i^e, e \in \text{adj}(v_i)$ sont dans V_T. En d'autres termes, on ajoute v_i dans V'_I si le chemin complet le représentant est présent dans V_T.

Prouvons maintenant que V'_I est bien une solution pour $\mathcal{I}$ telle que $|V'_I| \geq \left\lceil \frac{|V_T|-2|E_I|-1}{|V_I|^2} \right\rceil$. S'il existe $v_i, v_j \in V'_I$ tels que $\{v_i, v_j\} = e \in E$, alors v_i^e et v_j^e sont dans V_T. Ceci est impossible car $col(v_i^e) = col(v_j^e)$ et que les couleurs de V_T doivent être toutes distinctes pour être une solution au problème MAXIMUM GRAPH MOTIF avec un motif colorful. Par conséquent, le graphe V'_I est un ensemble indépendant. Le nombre de chemins entièrement présents dans V_T est égal à $\left\lceil \frac{|V_T|-2|E_I|-1}{|V_I|^2} \right\rceil$. En effet, en retirant $2|E_G|+1$ au nombre de sommets de la solution, on borne le nombre de sommets de type v_i^j (on rappelle que $|V| = 1 + 2|E_I| + |V_I|.|V_I|^2$). □

Prouvons enfin qu'un algorithme d'approximation pour MAXIMUM GRAPH MOTIF avec un ratio inférieur à $|V|^{\frac{1}{3}-\epsilon}, \epsilon > 0$ est improbable, sauf si $\mathsf{P} = \mathsf{NP}$.

Proposition 5.2.3. *Sauf si* $\mathsf{P} = \mathsf{NP}$*, on ne peut pas trouver de ratio d'approximation inférieur à* $|V|^{\frac{1}{3}-\epsilon}$ *pour le problème* MAXIMUM GRAPH MOTIF *pour n'importe quel* $\epsilon > 0$*, même lorsque le motif est colorful et* G *est un arbre où chaque couleur de* C *apparait au plus deux fois.*

Démonstration. Supposons qu'il existe un tel ratio r pour le problème MAXIMUM GRAPH MOTIF, permettant l'obtention d'une solution approximée $T_{APX} = (V_{T_{APX}}, E_{T_{APX}})$, qui comparée à une solution optimale $T_{OPT} = (V_{T_{OPT}}, E_{T_{OPT}})$ induit $|V_{T_{APX}}| \geq \frac{|V_{T_{OPT}}|}{r}$.

Par le Lemme 5.2.1

$$|V_{T_{OPT}}| \geq |V'_{I_{OPT}}|.|V_I|^2.$$

On a supposé

$$|V_{T_{APX}}| \geq \frac{|V_{T_{OPT}}|}{r}.$$

Donc,

$$|V_{T_{APX}}| \geq \frac{|V'_{I_{OPT}}|.|V_I|^2}{r}.$$

Par le Lemme 5.2.2,

$$|V'_{I_{APX}}| \geq \left\lceil \frac{|V_{T_{APX}}| - 2|E_I| - 1}{|V_I|^2} \right\rceil.$$

Ce qui mène à,

$$|V'_{I_{APX}}| \geq \left\lceil \frac{((|V'_{I_{OPT}}|.|V_I|^2)/r) - 2|E_I| - 1}{|V_I|^2} \right\rceil.$$

Comme $\frac{2|E_I|+1}{|V_I|^2} \leq 1$,

$$|V'_{I_{APX}}| \geq \left\lceil \frac{(|V'_{I_{OPT}}|.|V_I|^2)/r}{|V_I|^2} \right\rceil - 1.$$

Finalement,

$$|V'_{I_{APX}}| \geq \frac{|V'_{I_{OPT}}|}{r} - 1.$$

Ainsi, s'il existe un algorithme d'approximation avec un ratio r pour le problème MAXIMUM GRAPH MOTIF, nous obtenons un algorithme d'approximation de ratio r pour le problème MAXIMUM INDEPENDENT SET. Nous concluons la preuve en observant que $|V| = \mathcal{O}(|V_I|^3)$, et qu'il n'existe pas de ratio d'approximation inférieur à $|V_I|^{1-\epsilon}$ pour le problème MAXIMUM INDEPENDENT SET, $\forall \epsilon > 0$, sauf si P = NP [Zuc07]. □

5.2.2 Minimiser le nombre de substitutions

On s'intéresse maintenant à un cas particulier du problème GRAPH MOTIF WITH GAPS introduit sous le nom de MIN-SUBSTITUTE par Dondi *et al.* [DFV11], où la solution doit avoir une taille exactement égale à $|M|$ et où $k = |M| - s$, avec s un nombre soit fixé (problème de décision), soit à minimiser (problème de minimisation). Intuitivement, s correspond à un nombre de *substitutions*. On autorise que s couleurs du motif soient absentes, mais, la taille de la solution doit tout de même être égale à $|M|$ (voir Figure 5.4). Pour plus de commodité, on définit le problème (d'optimisation) sous un nom spécifique, proche de celui utilisé par [DFV11].

MINIMUM SUBSTITUTIONS GRAPH MOTIF

- **Entrée** : Un graphe $G = (V, E)$, un ensemble de couleurs C, une fonction $col : V \to C$, un multi-ensemble M sur C.
- **Sortie** : Un sous-ensemble $V_T \subseteq V$ tel que (i) $|V_T| = |M|$ et (ii) $G|V_T]$ soit connexe.
- **Mesure** : Le nombre de substitutions pour obtenir M depuis $col(V_T)$.

L'ensemble C est potentiellement plus grand que M car on doit pouvoir substituer des couleurs de M par des couleurs présentes dans $C \setminus M$.

Malheureusement, une nouvelle fois, même dans le cas restreint où le motif est colorful et G est un arbre de profondeur 2, on ne peut pas trouver d'algorithme d'approximation ayant un ratio inférieur à $c \log |V|$, pour une constante c. Le résultat étant négatif, il s'étend au cas où le motif est un multi-ensemble.

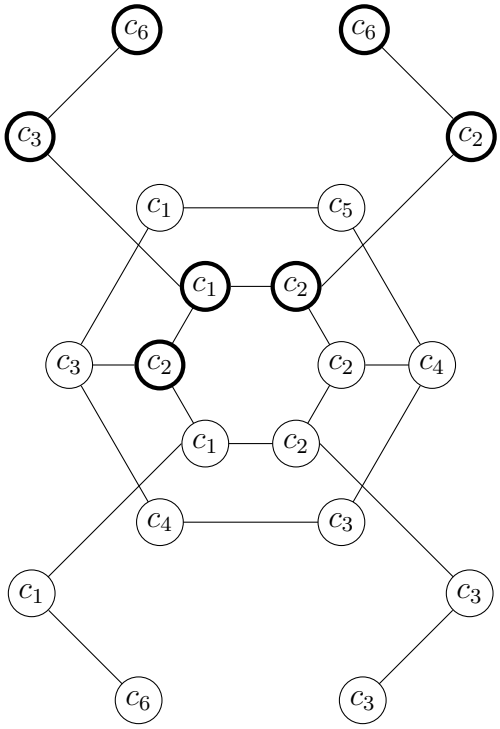


FIGURE 5.4 – Exemple fictif d'un graphe G coloré sur les sommets par l'ensemble de couleurs $C = \{c_i : 1 \leq i \leq 6\}$. On trouve en gras dans le réseau une solution possible $col(V_T) = \{c_1, c_2, c_2, c_2, c_3, c_6, c_6\}$ pour le problème MINIMUM SUBSTITUTIONS GRAPH MOTIF avec le motif $M = \{c_1, c_2, c_2, c_2, c_4, c_6, c_6\}$. On substitue une couleur (c_4 par un sommet de couleur c_3).

Pour obtenir une borne sur l'approximation de ce problème, on effectue une L-réduction depuis le problème MINIMUM SET COVER.

MINIMUM SET COVER

- **Entrée** : Un ensemble $X = \{x_1, x_2, \ldots, x_{|X|}\}$, une collection $\mathcal{S} = \{S_1, S_2, \ldots, S_{|\mathcal{S}|}\}$ de sous-ensembles de X.
- **Sortie** : Un sous-ensemble $\mathcal{S}' \subseteq \mathcal{S}$ tel que chaque élément de X apparaisse dans au moins un sous-ensemble de $\mathcal{S}'$.
- **Mesure** : La taille de $\mathcal{S}'$.

On note par $e(i, j)$ l'indice de X correspondant au j^{eme} élément de S_i. Voyons premièrement la construction de l'instance $\mathcal{I}' = (G, C, col, M)$ du problème MINIMUM SUBSTITUTIONS GRAPH MOTIF depuis une instance $\mathcal{I} = (X, \mathcal{S})$ du problème MINIMUM SET COVER.

Depuis une instance $\mathcal{I}$, on construit $G = (V, E)$ comme suit (voir aussi Figure 5.5).

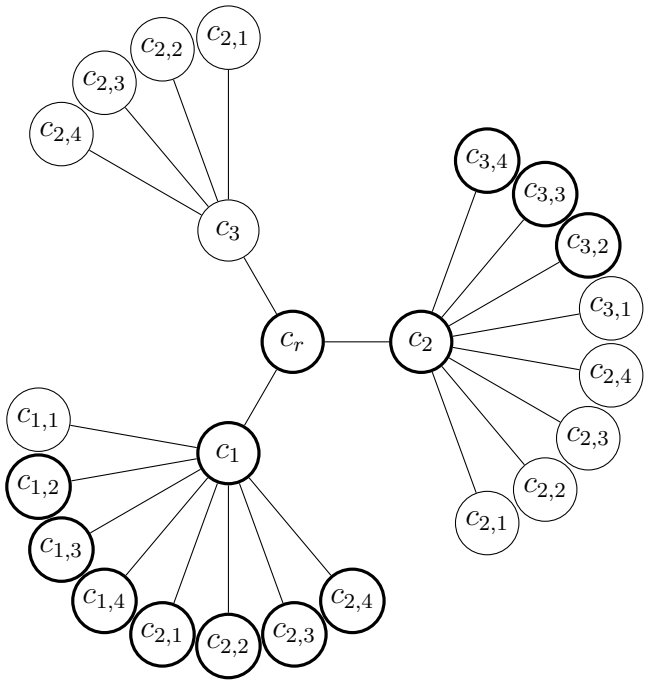
– $V = \{r\} \cup$

$\{v_i : 1 \leq i \leq |\mathcal{S}|\} \cup$

$\{v_{i,j,t} : 1 \leq i \leq |\mathcal{S}|, 1 \leq j \leq |S_i|, 1 \leq t \leq |\mathcal{S}| + 1\}$,

– $E = \{\{r, v_i\} : 1 \leq i \leq |\mathcal{S}|\} \cup$

$\{\{v_i, v_{i,j,t}\} : 1 \leq i \leq |\mathcal{S}|, 1 \leq j \leq |S_i|, 1 \leq t \leq |\mathcal{S}| + 1\}$.



$G = (V, E)$

FIGURE 5.5 – Exemple de construction d'une instance de MINIMUM SUBSTITUTIONS GRAPH MOTIF depuis une instance de MINIMUM SET COVER telle que $X = \{x_1, x_2, x_3\}$ et $\mathcal{S} = \{\{x_1, x_2\}, \{x_2, x_3\}, \{x_2\}\}$. Pour des raisons de lisibilité, on indique seulement la couleur de chaque sommet (et non leur nom). Le motif associé à ce graphe est $M = \{c_r\} \cup \{c_{k,t} : 1 \leq k \leq 3, 1 \leq t \leq 4\}$. Une solution possible (avec deux substitutions) est illustrée en gras.

De manière informelle, r est la racine d'un arbre ayant $|\mathcal{S}|$ fils, correspondant aux sous-ensembles de $\mathcal{S}$. Chaque fils $v_i, 1 \leq i \leq |\mathcal{S}|$, possède $(|\mathcal{S}|+1)|S_i|$ fils, correspondant à $|\mathcal{S}| + 1$ copies de chaque élément contenu dans S_i.

L'ensemble de couleurs est $C = \{c_r\} \cup \{c_i : 1 \leq i \leq |\mathcal{S}|\} \cup \{c_{k,t} : 1 \leq k \leq |X|, 1 \leq t \leq |\mathcal{S}|+1\}$. La fonction de coloration des sommets est telle qu'on donne une couleur unique à la racine $col(r) = c_r$. Chaque sommet v_i reçoit une couleur unique correspondant à un sous-ensemble de $\mathcal{S}$, $col(v_i) = c_i, \forall 1 \leq i \leq |\mathcal{S}|$. Chaque sommet $v_{i,j,t}$ reçoit la couleur correspondant à la copie de l'élément qu'il représente, c'est-à-dire $col(v_{i,j,t}) = c_{e(i,j),t}$.

Enfin, le motif est défini par $M = \{c_r\} \cup \{c_{k,t} : 1 \leq k \leq |X|, 1 \leq t \leq |\mathcal{S}| + 1\}$. On remarque en particulier que les couleurs $\{c_i : 1 \leq i \leq |\mathcal{S}|\}$ ne sont pas présentes dans le motif (qui est colorful par construction).

Voyons maintenant comment obtenir une solution pour $\mathcal{I}'$ depuis une solution pour $\mathcal{I}$.

Lemme 5.2.4. *Depuis une solution $\mathcal{S}'$ pour une instance $\mathcal{I}$ du problème* MINIMUM SET COVER, *il existe une solution pour l'instance $\mathcal{I}'$ de* MINIMUM SUBSTITUTIONS GRAPH MOTIF *avec $|\mathcal{S}'|$ substitutions.*

Démonstration. Soit $\mathcal{S}' \subseteq \mathcal{S}$ une solution pour $\mathcal{I}$. Étant donné un ordre total sur les sous-ensembles de $\mathcal{S}$, pour chaque $1 \leq k \leq |X|$, on note par S^k_{min} le sous-ensemble tel que (i) $S^k_{min} \in \mathcal{S}'$ et (ii) S^k_{min} est le premier sous-ensemble de $\mathcal{S}'$ contenant x_k. De plus, pour chaque S_i, on note f_i le plus petit indice j de $v_{i,j,t}$ tel que $S_i = S^{e(i,j)}_{min}$.

On construit une solution $T = (V_T, E_T)$ pour $\mathcal{I}'$ comme suit. L'ensemble des sommets de T est $V_T = \{r\} \cup \{v_i : S_i \in \mathcal{S}'\} \cup \{v_{i,j,t} : S_i = S^{e(i,j)}_{min}, j = f_i, 2 \leq t \leq |\mathcal{S}| + 1\} \cup \{v_{i,j,t} : S_i = S^{e(i,j)}_{min}, j \neq f_i, 1 \leq t \leq |\mathcal{S}| + 1\}$. Intuitivement, on place dans la solution la racine, l'ensemble des sommets représentant les S_i appartenant à $\mathcal{S}'$ ainsi que les $|\mathcal{S}| + 1$ copies de chaque sommet représentant un x_k (celle qui appartient au sous-ensemble ayant l'indice minimal dans la solution), sauf pour l'élément x_k de X qui porte le plus petit indice dans S^k_{min}, où l'on ajoute seulement $|\mathcal{S}|$ copies dans la solution.

Il est clair que $G[V_T]$ est connexe car les sommets $v_{i,j,t}$ sont dans la solution si et seulement si le sommet v_i l'est également. Enfin, un sommet v_i est dans la solution s'il existe un k tel que $S_i = S^k_{min}$. Il existe donc un f_i pour lequel seulement $|\mathcal{S}|$ copies de $v_{i,f_i,t}$ sont dans la solution. Par conséquent, par construction, la couleur $c_{e(i,f_i),1}$, présente dans le motif, est substituée dans la solution par c_i. Il y a donc bien au total $|\mathcal{S}'|$ substitutions effectuées, car les autres couleurs du motif sont présentes dans la solution. □

Voyons maintenant comment obtenir une solution pour $\mathcal{I}$ depuis une solution pour $\mathcal{I}'$.

Lemme 5.2.5. *Depuis une solution pour l'instance $\mathcal{I}'$ de* MINIMUM SUBSTITUTIONS GRAPH MOTIF *avec au plus s substitutions, il existe une solution pour l'instance $\mathcal{I}$ de* MINIMUM SET COVER *de taille au plus s.*

Démonstration. Soit $T = (V_T, E_T)$ une solution pour $\mathcal{I}'$ telle que l'on retrouve M depuis $col(V_T)$ avec au plus s substitutions. Nous pouvons supposer que $s < |\mathcal{S}| + 1$, car sinon, prendre $\mathcal{S}' = \mathcal{S}$ est une solution de taille satisfaisante.

La solution pour $\mathcal{I}$ est alors construite comme suit : $\mathcal{S}' = \{S_i : v_i \in V_T\}$. Si pour un $1 \leq k \leq |X|$ il n'y a aucune des couleurs de $\{c_{k,t} : 1 \leq t \leq |\mathcal{S}| + 1\}$ présente dans la solution, alors cela signifie que ces $|\mathcal{S}|+1$ couleurs ont toutes été substituées, ce qui est en contradiction avec le nombre s de substitutions supposées. Par conséquent, pour chaque

$1 \leq k \leq |X|$, il existe au moins un sommet portant une couleur de $\{c_{k,t} : 1 \leq t \leq |\mathcal{S}|+1\}$, donc, comme la solution est connexe, tous les éléments de X sont couverts par $\mathcal{S}'$.

Enfin, la taille de $\mathcal{S}'$ est bornée par s car, comme leurs couleurs ne sont pas dans le motif, il y a au plus s sommets v_i dans V_T.

□

Cette construction et ces deux lemmes mènent au résultat principal sur l'approximation du problème MINIMUM SUBSTITUTIONS GRAPH MOTIF.

Proposition 5.2.6. *Sauf si* P = NP, *il n'existe pas d'algorithme d'approximation ayant un ratio inférieur à* $c \log |V|$, c *une constante, pour le problème* MINIMUM SUBSTITUTIONS GRAPH MOTIF, *même lorsque le motif est colorful et* G *est un arbre de profondeur 2.*

Démonstration. La preuve découle directement des Lemmes 5.2.4 et 5.2.5 et du fait qu'il n'existe pas d'algorithme d'approximation ayant un ratio inférieur à $c \log |X|$ pour le problème MINIMUM SET COVER, sauf si P = NP [RS97]. On remarque que le paramètre est exactement identique entre les deux instances $\mathcal{I}$ et $\mathcal{I}'$, c'est donc bien une L-réduction. □

En corollaire de la Proposition 5.2.6, on remarque que la réduction utilisée est également une FPT-réduction. Comme le problème MINIMUM SET COVER est W[2]-Difficile lorsqu'il est paramétré par le nombre de sous-ensembles de la solution [Nie06], le problème MINIMUM SUBSTITUTIONS GRAPH MOTIF est également W[2]-Difficile lorsqu'il est paramétré par le nombre de substitutions. On remarque que l'on savait déjà le problème W[1]-Difficile avec ce même paramètre puisque le contraire impliquerait un algorithme avec une complexité $f(s).|V|^c$, avec s le nombre de substitutions et c une constante. Pour $s = 0$, on obtiendrait un algorithme polynomial pour résoudre le problème EXACT GRAPH MOTIF, que l'on sait NP-Complet.

5.3 Utilisation de modules pour GRAPH MOTIF

Nous proposons dans cette section une légère modification du problème EXACT GRAPH MOTIF, où, le sous-ensemble de sommets devant respecter les couleurs d'un motif doit être un module du graphe, et non plus être simplement connexe. Après avoir brièvement donné les notions essentielles sur les modules, nous donnons quelques pistes sur la justification de leurs utilisations. Enfin, si le problème reste NP-Difficile, les outils fournis par les modules permettent l'obtention d'algorithmes efficaces simplement. Ce problème étant encore en cours d'étude, les résultats de cette section (certains obtenus en collaboration avec Romeo Rizzi) n'ont pas été publiés.

5.3.1 Définitions autour des modules

Dans un graphe non-orienté $G = (V, E)$ considéré sans boucle, on dit qu'un sommet $x \in V$ *sépare* deux sommets $u \in V$ et $v \in V$ si et seulement si $\{x, u\} \in E$ et $\{x, v\} \notin E$. Un *module* $\mathcal{M}$ d'un graphe G est un ensemble de sommets qui ne sont séparés par aucun sommet de $V \setminus \mathcal{M}$. On peut également dire qu'un module $\mathcal{M}$ est tel que $\forall x \notin \mathcal{M}, \forall u, v \in \mathcal{M}, \{x, u\} \in E \Leftrightarrow \{x, v\} \in E$ [MR84] (voir Figure 5.6). Autrement dit, un module $\mathcal{M}$ est un ensemble de sommets de V ayant une relation uniforme avec le reste du graphe, c'est-à-dire avec $V \setminus \mathcal{M}$ (voir Figure 5.7). L'ensemble V est appelé module trivial, tout comme chaque singleton $\{u\}$, où $u \in V$.

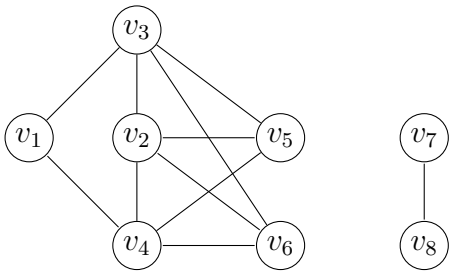
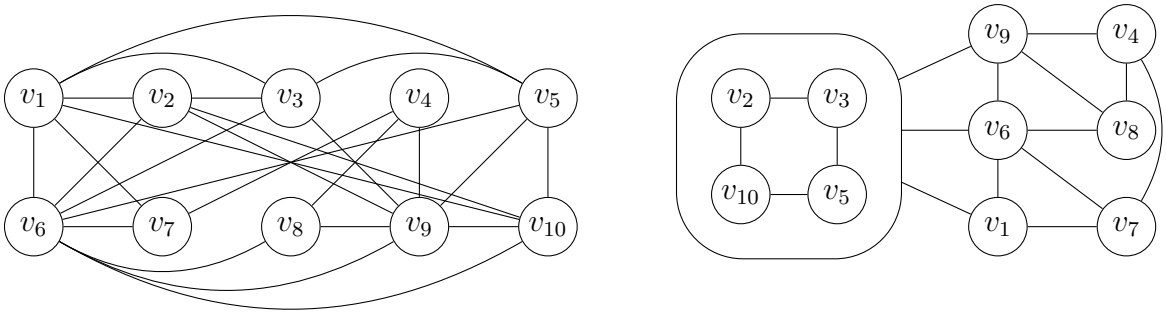


FIGURE 5.6 – Graphe d'exemple dont les modules sont : l'ensemble des sommets V du graphe, chaque singleton $\{v\}, \forall v \in V$, les deux composantes connexes $\{v_1, v_2, v_3, v_4, v_5, v_6\}$ et $\{v_7, v_8\}$, ou encore les ensembles $\{v_3, v_4\}, \{v_5, v_6\}, \{v_1, v_2, v_5, v_6\}$.



$a)$

FIGURE 5.7 – En $a)$, un graphe quelconque. En $b)$, le module $\{v_2, v_3, v_5, v_{10}\}$ est mis en évidence. Le dessin du graphe est plus clair. Exemple tiré de [Mon03].

Avant de donner la définition de modules spécifiques, on dit que deux modules A et B *se chevauchent* si (i) $A \cap B \neq \emptyset$, (ii) $A \setminus B \neq \emptyset$ et (iii) $B \setminus A \neq \emptyset$. Selon [CHM81], si deux modules A et B se chevauchent, alors $A \cap B$, $A \cup B$ et $(A \cup B) \setminus (A \cap B)$ sont également des modules.

Ceci nous permet de définir un certain type de modules, que l'on dit *forts*. Un module est fort si aucun autre module ne le chevauche (sinon il est dit *faible*). Par conséquent, deux modules forts sont soit inclus l'un dans l'autre, soit ont une intersection vide.

Un module $\mathcal{M} \subset S$ est dit *maximal* pour un ensemble de sommets S (par défaut l'ensemble des sommets V) s'il n'existe aucun module $\mathcal{M}'$ tel que $\mathcal{M} \subset \mathcal{M}' \subset S$. Autrement dit, le seul module contenant le module maximal $\mathcal{M}$ est S.

Un module peut être de trois types :

- il est *parallèle* si le sous-graphe induit par les sommets du module n'est pas connexe (c'est une composition parallèle de ses composantes connexes),
- il est *série* si le complémentaire du sous-graphe induit par les sommets du module n'est pas connexe (c'est une composition série des composantes connexes de son complément),
- il est *premier* si le sous-graphe induit par les sommets du module ainsi que le complémentaire du sous-graphe induit par les sommets du module sont tous les deux connexes.

L'ordre d'inclusion des modules forts maximaux définit l'*arbre de décomposition modulaire* $\mathcal{T}(G)$ de G, contenant l'ensemble des modules forts du graphe. L'arbre $\mathcal{T}(G)$ peut être construit récursivement du haut vers le bas, en itérant sur le graphe induit par le module fort considéré. La racine de cet arbre est l'ensemble des sommets du graphe V, tandis que les feuilles sont les singletons $\{u\}$, $\forall u \in V$.

Chaque sommet de $\mathcal{T}(G)$ porte une étiquette représentant le type du module fort, *parallèle*, *série* ou *premier*. Les fils d'un sommet interne $\mathcal{M}$ sont les sous-modules maximaux de $\mathcal{M}$ (ils sont donc disjoints). La Figure 5.8 donne un exemple de la construction de $\mathcal{T}(G)$ depuis un graphe G. L'arbre de décomposition modulaire peut être obtenu avec un algorithme ayant une complexité linéaire, comme par exemple celui décrit dans [HMP04].

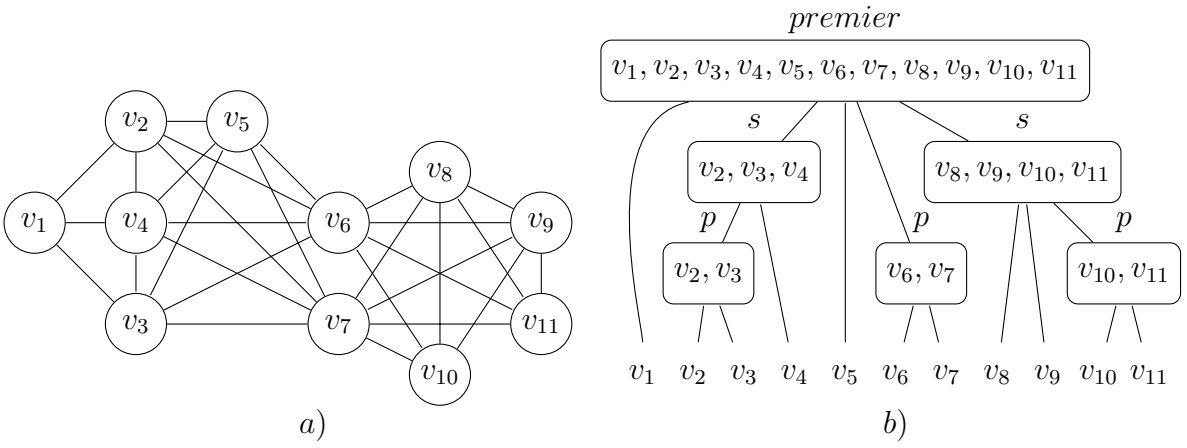


FIGURE 5.8 – Un graphe exemple en a) et son arbre de décomposition modulaire associé en b). Les sommets de l'arbre sont soit séries (s), parallèles (p), soit premiers ($premier$) ou des feuilles.

Nous pouvons maintenant introduire une propriété essentielle de $\mathcal{T}(G)$.

Théorème 5.3.1. *([CHM81]) Un module de G est soit un sommet de $\mathcal{T}(G)$, soit l'union de fils (de profondeur 1) d'un sommet série ou parallèle de $\mathcal{T}(G)$.*

On peut voir que les modules forts agissent comme des générateurs des modules de G – l'ensemble des modules de G peut être obtenu depuis l'arbre $\mathcal{T}(G)$. Un point décisif à observer est qu'il y a potentiellement un nombre exponentiel de modules dans un graphe (par exemple, la clique K_n contient 2^n modules), mais la taille de $\mathcal{T}(G)$ est $\mathcal{O}(n)$ (plus précisément, $\mathcal{T}(G)$ contient moins de $2n$ sommets puisqu'il comporte n feuilles et aucun sommet avec un seul fils). Par conséquent, on peut représenter la famille de taille exponentielle des modules de G par un arbre de taille linéaire .

5.3.2 Quand les modules rejoignent GRAPH MOTIF

Dans la suite de la section, on étudie le comportement algorithmique du problème de la recherche de motif sans topologie, où la demande de simple connexité est remplacée par la modularité. En suivant la définition du problème EXACT GRAPH MOTIF, on obtient le problème suivant.

MODULE GRAPH MOTIF

• **Entrée** : Un graphe $G = (V, E)$, un ensemble de couleurs C, une fonction $col : V \to C$, un multi-ensemble M de taille k sur C.

• **Sortie** : Un sous ensemble $V_T \subseteq V$ tel que (i) V_T soit un module de G et (ii) $col(V_T) = M$.

Nous lions la demande de modularité à la recherche d'un motif. La définition d'un module implique que tous les sommets de celui-ci aient une relation uniforme avec l'ensemble des sommets extérieurs au module. Les sommets d'un module ne peuvent pas être distingués de l'extérieur, ils ont le même comportement avec tous les autres sommets du graphe.

Les auteurs de [AA03] définissent un module biologique comme un ensemble d'éléments effectuant une fonction séparable du reste du graphe. De manière analogue, les auteurs de [RSM+02] décrivent un module biologique comme un ensemble de plusieurs éléments ayant une tâche identifiable, séparable des fonctions des autres modules.

De plus, dans [CBB+10], il est montré que des gènes ayant un voisinage similaire ont de fortes chances de faire partie d'un même processus biologique. Il est donc possible que l'ensemble des sommets d'un module d'un graphe représentant un réseau biologique porte une fonction biologique commune. Par ailleurs, les auteurs de [SSR+03] décrivent les modules dans les réseaux de régulation de gènes comme des groupes de gènes obéissant aux mêmes régulations, et qui, par conséquent, ne peuvent pas être distingués du reste du réseau.

En outre, après avoir utilisé les modules dans un cadre légèrement différent afin de prédire plus finement les résultats de découverte d'interactions entre des protéines, Gagneur *et al.* [GKBC04] indiquent que les modules d'un graphes peuvent rejoindre

les modules biologiques et envisagent la décomposition modulaire comme un outil général pour l'analyse de différents réseaux biologiques, sous différentes représentations (graphes orientés, hypergraphes...).

Néanmoins, il n'existe pas de définition précise de ce que doit (ou devrait) être un module dans un réseau biologique [AA03]. Nous pensons donc que l'approche utilisant les modules d'un graphe est complémentaire aux définitions précédentes de modules biologiques (une occurrence simplement connectée, une occurrence très connexe proche d'une clique...).

5.3.3 Difficulté du problème

Malheureusement, il s'avère que le problème MODULE GRAPH MOTIF est NP-Difficile même sous de très fortes conditions, à savoir quand G est une collection de chemins de taille trois et que le motif est colorful.

On remarque que dans ces mêmes conditions, le problème EXACT GRAPH MOTIF peut se résoudre de manière polynomiale. Ceci est dû au fait que la solution dans ce dernier doit absolument être connexe, ce qui n'est pas le cas pour le problème MODULE GRAPH MOTIF.

Proposition 5.3.2. *Le problème* MODULE GRAPH MOTIF *est* NP*-Complet, même quand le motif est colorful et G est une collection de chemins de taille 3.*

Démonstration. Le problème MODULE GRAPH MOTIF appartient à NP car étant donné un ensemble $V' \subseteq V$, on peut vérifier en temps polynomial si V' est un module et si toutes les couleurs de C apparaissent exactement une fois dans V'. Pour prouver que le problème est NP-Difficile, on propose une réduction depuis le problème EXACT COVER BY 3-SETS (X3C). Ce cas particulier du problème SET COVER est connu pour être NP-Complet [GJ79].

X3C

• **Entrée** : Un entier $q \geq 1$, un ensemble $X = \{x_1, x_2, \ldots, x_{3q}\}$, une collection $\mathcal{S} = \{S_1, \ldots, S_{|\mathcal{S}|}\}$ de triplets de X.

• **Sortie** : Un sous-ensemble $\mathcal{S}' \subseteq \mathcal{S}$ tel que chaque élément de X apparaisse dans exactement un membre de $\mathcal{S}'$.

La taille de X est forcément un multiple de trois car une solution est un ensemble de triplets où chaque élément de X doit apparaitre exactement une fois.

Définissons maintenant la construction d'une instance $\mathcal{I}' = (G, C, col)$ du problème MODULE GRAPH MOTIF depuis une instance $\mathcal{I} = (X, \mathcal{S})$ du problème X3C (voir aussi Figure 5.9). Le graphe $G = (V, E)$ est construit comme suit :

- $V = \{v_i^j : 1 \leq i \leq |\mathcal{S}|, x_j \in S_i\}$,
- $E = \{\{v_i^1, v_i^2\} \cup \{v_i^2, v_i^3\} : 1 \leq i \leq |\mathcal{S}|\}$.

Intuitivement, G est une collection de $|\mathcal{S}|$ chemins comportant trois sommets (on rappelle que pour tous les $1 \leq i \leq |\mathcal{S}|, |S_i| = 3$).

L'ensemble des couleurs utilisé est $C = \{c_i : 1 \leq i \leq |X|\}$. La coloration des sommets de G est telle que $col(v_i^j) = c_j$. Intuitivement, chaque sommet porte la couleur de l'élément de X qu'il représente. Comme on cherche un motif colorful, une solution doit porter toutes les couleurs de C. Cette construction s'effectue clairement en temps polynomial par rapport à $\mathcal{I}$.

Prouvons maintenant que s'il existe une solution pour une instance $\mathcal{I}$ de X3C, alors il existe une solution pour l'instance $\mathcal{I}'$ de MODULE GRAPH MOTIF. Soit $\mathcal{S}' \subseteq \mathcal{S}$ une solution pour $\mathcal{I}$. On construit une solution V_T pour $\mathcal{I}'$ telle que $V_T = \{v_i^j : S_i \in \mathcal{S}', x_j \in S_i\}$. Intuitivement, on ajoute l'ensemble des chemins correspondants aux triplets choisis dans la solution. L'ensemble V_T est bien un module et par définition d'une solution pour $\mathcal{I}$, chaque couleur de $\{c_i : 1 \leq i \leq |X|\}$ apparaît exactement une fois dans V_T.

Prouvons maintenant l'inverse, c'est-à-dire qu'il existe une solution pour l'instance $\mathcal{I}$ de X3C s'il existe une solution pour l'instance $\mathcal{I}'$ de MODULE GRAPH MOTIF. On remarque premièrement que comme $q \geq 1$, nous avons $|X| \geq 3$ et donc $C \geq 3$. Un module de taille supérieure ou égale à trois dans une collection de chemins de taille trois est forcément une union de chemins de taille trois. En effet, supposons par contradiction un module $\mathcal{M}$ de taille supérieure à trois n'étant pas une union de chemins de taille trois. Il existe donc un sommet $u \in \mathcal{M}$ tel qu'au moins un de ses voisins $v \in N(u)$ n'appartienne pas à $\mathcal{M}$. Alors, v *sépare* u d'un autre sommet de $\mathcal{M}$ et donc $\mathcal{M}$ n'est pas un module. On construit $\mathcal{S}' = \{S_i : v_i^j \in V_T\}$. Comme la solution V_T est une union de chemins de taille trois, chaque triplet S_i est soit entièrement choisi dans la solution, soit absent. Comme c'est une solution, les couleurs de V_T sont présentes exactement une fois. Par conséquent, chaque élément de X apparaît exactement une fois dans $\mathcal{S}'$. □

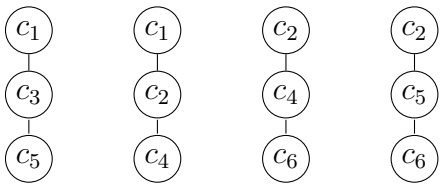


FIGURE 5.9 – Le graphe G construit depuis $X = \{x_1, x_2, \ldots, x_6\}$ (donc $q = 2$) et $\mathcal{S} = \{\{x_1, x_3, x_5\}, \{x_1, x_2, x_4\}, \{x_2, x_4, x_6\}, \{x_2, x_5, x_6\}\}$ (on indique uniquement les couleurs des sommets). Par construction, l'ensemble de couleurs utilisé et demandé dans la solution est $C = \{c_1, c_2, \ldots, c_6\}$.

5.3.4 Des algorithmes pour le problème de décision

Même si le problème est rapidement difficile, l'utilisation de modules permet parfois des algorithmes de complexité paramétrée efficaces. C'est donc un des avantages de l'utilisation de modules.

Plus précisément, on montre que le problème est dans la classe FPT lorsque le paramètre est la taille de la solution, avec une complexité plus faible que pour le problème EXACT GRAPH MOTIF quand le motif est un multi-ensemble. En outre, le problème reste dans la classe FPT lorsqu'une liste de couleurs est associée à chaque sommet du graphe.

On observe dans un premier temps qu'en demandant un module fort plutôt qu'un module quelconque en tant que solution pour le problème MODULE GRAPH MOTIF, alors l'algorithme est clairement linéaire puisqu'il suffit de parcourir $\mathcal{T}(G)$ et tester si l'ensemble de couleurs de chaque module fort est égal au motif.

On montre maintenant que l'on peut obtenir un algorithme ayant une complexité en temps de $\mathcal{O}^*(2^k)$ où k est la taille de la solution pour le problème MODULE GRAPH MOTIF, même lorsque le motif est un multi-ensemble. Ce résultat se contraste avec le fait qu'il n'existe pour l'instant pas d'algorithme ayant une complexité inférieure à $\mathcal{O}^*(4^k)$ pour le problème EXACT GRAPH MOTIF si le motif est un multi-ensemble (Proposition 4.3.2).

Proposition 5.3.3. *On peut résoudre le problème* MODULE GRAPH MOTIF *par un algorithme ayant une complexité en temps de $\mathcal{O}(2^k|V|^2)$ et une complexité en espace de $\mathcal{O}(2^k|V|)$, où k est la taille du motif et de la solution.*

Démonstration. Comme $|M| = k$, on remarque qu'il existe au plus 2^k multi-ensembles M' différents tels que $M' \subseteq M$.

Dans un premier temps, on construit $\mathcal{T}(G)$, l'arbre de décomposition modulaire de G. L'algorithme suivant est effectué pour chaque sommet $\mathcal{M}$ de $\mathcal{T}(G)$.

On commence par tester si les couleurs de $\mathcal{M}$ sont exactement égales à M. Si c'est le cas, l'algorithme est terminé. Sinon, si $\mathcal{M}$ est un sommet étiqueté série ou parallèle, un module peut être une union de ses fils. Selon un ordre arbitraire de ses t fils, on note $\mathrm{Child}(\mathcal{M})[i]$ le i^{eme} fils de $\mathcal{M}$. On supprime tous les fils $\mathcal{M}'$ de $\mathcal{M}$ tels que $col(\mathcal{M}') \not\subset M$, où $col(\mathcal{M}')$ représente l'ensemble des couleurs des sommets de $\mathcal{M}'$. En effet, un tel fils ne pourrait jamais être présent dans une solution pour M. On remarque que l'ensemble des couleurs de chaque fils correspond à un multi-ensemble $M' \subseteq M$.

Comme n'importe quelle union des fils de $\mathcal{M}$ est un module, c'est donc une solution potentielle. On propose de tester par programmation dynamique si l'une de ces unions correspond à une solution pour M. On définit une table $D(i, M')$, pour $0 \leq i \leq t$ et $M' \subseteq M$. Par conséquent, D contient $t + 1$ lignes et 2^k colonnes. On remplit cette table de la manière suivante :

$$D(0, M') = Vrai \text{ si } M' = \{0, \ldots, 0\},\ Faux \text{ sinon,}$$
$$D(i, M') = D(i-1, M') \vee D(i-1, M' \setminus col(\mathrm{Child}(\mathcal{M})[i])) \text{ si } i \leq t, M' \subseteq M.$$

On répond $Vrai$ seulement si $D(t, M) = Vrai$.

De manière informelle, la première partie du calcul de $D(i, M')$ ignore le i^{eme} fils de $\mathcal{M}$ tandis que la seconde partie ajoute ce fils dans la solution potentielle.

La complexité en temps et en espace de la programmation dynamique est de $\mathcal{O}(2^k|V|)$, car D contient au plus $2^k|V|$ cases et que le temps de calcul pour chaque case de cette table est constant. Par conséquent, comme la programmation dynamique est lancée au pire sur chaque sommet de $\mathcal{T}(G)$, la complexité totale en temps de l'algorithme est bien de $\mathcal{O}(2^k|V|^2)$.

Il reste à montrer que la programmation dynamique est correcte. Supposons qu'il existe un module $\mathcal{M}'$ tel que $col(\mathcal{M}') = M$. Alors, soit $\mathcal{M}'$ est un module fort étant représenté dans un sommet de $\mathcal{T}(G)$, soit c'est une union de j modules $\mathcal{M}'_1, \mathcal{M}'_2, \ldots, \mathcal{M}'_j$ fils d'un module $\mathcal{M}$. Alors, $M \backslash \{\{col(\mathcal{M}'_1) \cup \{col(\mathcal{M}'_2)\} \cup \cdots \cup \{col(\mathcal{M}'_j)\}\}\} = \{0, 0, \ldots, 0\}$. Donc $D(t, \mathcal{M}) = Vrai$.

Inversement, s'il existe $\mathcal{M}$ tel que $D(t, \mathcal{M}) = Vrai$, alors il existe une union des fils de $\mathcal{M}$ telle que leur ensemble de couleur est égal à M. □

Corollaire 5.3.4. *Si le nombre de couleurs différentes dans le motif est borné, alors* MODULE GRAPH MOTIF *peut-être résolu par un algorithme polynomial.*

Démonstration. On observe que, par définition du motif M, pour chaque couleur $c \in C, occ_M(c) \leq k$. Par conséquent, le nombre de multi-ensembles M' tels que $M' \subseteq M$ est inférieur à $k^{|C|}$. Alors, la complexité de l'algorithme de la Proposition 5.3.3 est bornée par $\mathcal{O}(k^{|C|}|V|^2)$, qui est polynomial si le nombre de couleurs est borné. □

Ce corollaire est plutôt étonnant et montre une différence fondamentale avec le problème EXACT GRAPH MOTIF. En effet, on rappelle que ce dernier est NP-Complet, même lorsque le motif utilise seulement deux couleurs différentes [FFHV07].

On montre maintenant que même si un ensemble de couleurs est associé à chaque sommet du graphe, le problème reste de complexité paramétrée. Ce problème correspond au problème LIST-COLORED GRAPH MOTIF, où la contrainte de connexité est remplacée par celle de modularité. Nous avions déjà vu qu'associer une liste de couleurs à chaque sommet d'un graphe était pertinent dans le cadre d'applications biologiques.

LIST-COLORED MODULE GRAPH MOTIF

• **Entrée** : Un graphe $G = (V, E)$, un entier k, un ensemble de couleurs C, un multi-ensemble M sur C, une fonction $col : V \to 2^C$ donnant pour chaque sommet de V l'ensemble de couleurs associé.

• **Sortie** : Un sous ensemble $V_T \subseteq V$ tel que (i) $|V_T| = k$, (ii) V_T soit un module de G et (iii) il existe une bijection $f : V_T \to M$ telle que $\forall v \in V_T, f(v) \in col(v)$.

Proposition 5.3.5. *Le problème* LIST-COLORED MODULE GRAPH MOTIF *est dans la classe* FPT.

Démonstration. Dans un premier temps, on construit $\mathcal{T}(G)$, l'arbre de décomposition modulaire de G. L'algorithme suivant est effectué pour chaque sommet $\mathcal{M}$ de $\mathcal{T}(G)$.

Si $\mathcal{M}$ contient moins de k sommets, on teste s'il existe une bijection entre les couleurs de $\mathcal{M}$ et M. Pour cela, on effectue toutes les combinaisons possibles, c'est-à-dire au pire c^k, où c est le nombre de couleurs différentes dans M, donc $c \leq k$.

Dans la suite, on considère donc que $\mathcal{M}$ contient plus de k sommets. S'il est étiqueté premier, on l'ignore car il ne peut pas être solution à un motif de taille k. Sinon, il est étiqueté série ou parallèle et une union de ses fils peut-être une solution. On montre maintenant que le nombre de solutions possibles est exponentiel uniquement selon k et qu'il est donc possible de toutes les tester.

Pour cela, on borne dans un premier temps le nombre de fils considérés pour $\mathcal{M}$. Il y a au plus k sommets dans chacun des fils de $\mathcal{M}$ (sinon ce fils ne peut jamais être dans une solution). Dans chaque fils de $\mathcal{M}$, il existe au plus 2^c ensembles de couleurs différents associés à chaque sommet. Comme on considère au plus k sommets dans chaque fils de $\mathcal{M}$, il y a donc au plus $(2^c)^k$ fils de $\mathcal{M}$ différents. Un même fils de $\mathcal{M}$ ne peut être répété que k fois (sinon, les occurrences suivantes ne peuvent pas faire partie d'une solution pour M). Par conséquent, il y a au plus $k(2^c)^k$ fils à considérer pour $\mathcal{M}$.

Nous avons donc borné le nombre de fils de $\mathcal{M}$. Nous devons maintenant choisir l'union potentielle des fils de $\mathcal{M}$ étant dans la solution – il faut en choisir i parmi les $k(2^c)^k$, avec i allant de 1 à k. Ceci est borné par $(k(2^c)^k)^{k+1}$. Enfin, pour chaque ensemble de fils choisi, il y a c couleurs possibles pour les sommets (dont le nombre est borné par k), menant à au plus c^k tests. Au final, la complexité de l'algorithme dépend uniquement de k. □

Il est évident que la complexité de l'algorithme de la Proposition 5.3.5 n'est pas envisageable pour une implémentation. Il est certainement possible d'améliorer cette complexité en utilisant la technique de détection de monômes multilinéaires.

5.3.5 Problèmes ouverts et discussions

Il est certain que l'imprécision dans les données biologiques implique que la recherche exacte d'un module est beaucoup trop restrictive pour être mise en pratique. On rappelle en effet que pour être dans un module, tous les sommets de ce module doivent avoir le même voisinage extérieur. Ainsi, un seul faux positif ou un seul faux négatif peut empêcher la détection d'un module. L'ajout de flexibilité comme dans les problèmes vus précédemment dans ce chapitre semble donc indispensable.

Flexibilité sur le résultat demandé

On remarque premièrement qu'un algorithme de complexité paramétrée pour le problème cherchant à minimiser le nombre de délétions est facilement envisageable. Il suffit en effet d'exécuter l'algorithme de la Proposition 5.3.3 pour tous les motifs $M' \subseteq M$ tels que $|M'| = |M| - d$, où d est le nombre d'occurrences de M absentes de M',

allant de 0 à N_{del} (N_{del} étant le nombre maximal de délétions possibles). Il existe $\binom{|M|}{|M|-d}$ motifs M' de cette taille – c'est-à-dire un nombre exponentiel selon la taille du motif uniquement.

Concernant les insertions, on pourrait envisager l'insertion de couleurs qui n'étaient pas demandées dans le motif comme pour GRAPH MOTIF WITH GAPS. Mais, on peut également envisager l'insertion de sommets n'appartenant pas au module (toujours dans le but de prendre en compte l'imprécision des données biologiques).

Betlzer *et al.* [BFKN08] étudient le problème EXACT GRAPH MOTIF, où la solution doit en plus être 2-connexe (c'est-à-dire que la suppression d'un sommet de l'occurrence laisse cette dernière connexe). Malheureusement, le problème devient alors W[1]-Difficile, même si paramétré par la taille de la solution.
Quelle est la complexité du problème si la treewidth de la solution est bornée ? Si elle est bornée par 1, la solution doit être un arbre. Si elle est bornée par 2, la solution doit être un graphe série-parallèle. De manière générale, que pourraient être des contraintes de connexité intéressantes d'un point de vue biologique ?

Flexibilité sur la propriété de module

Quelques travaux ont déjà tenté d'utiliser les modules dans les réseaux biologiques. Une interprétation de leur comportement est faite dans [GKBC04, DMER09].

- Un module fort $\mathcal{M}$ étiqueté parallèle se comporte comme une alternative entre ses fils. Chacun d'entre eux peut remplacer $\mathcal{M}$ avec succès dans n'importe quelle opération impliquant $\mathcal{M}$.
- Un module fort $\mathcal{M}$ étiqueté série demande que tous ses fils soient impliqués pour le remplacer dans une opération impliquant $\mathcal{M}$.
- Il n'y a pas d'interprétation biologique raisonnable pour un module fort étiqueté premier. En effet, un tel sommet peut correspondre à des graphes très différents les uns des autres.

C'est en partant de ce manque d'interprétation pour les modules premiers que Del Mondo *et al.* [DMER09] tentent de raffiner l'arbre de décomposition modulaire, avec l'*homogeneous decomposition*, introduisant deux nouveaux types de sommets pour les modules premiers. Est-il possible d'utiliser cette nouvelle structure dans le cadre du problème MODULE GRAPH MOTIF ?

Un *split* [Cun82] de $G = (V, E)$ est une partition de V en deux sous-ensembles disjoints V_1 et V_2 telle que :

- $|V_1| > 1$ et $|V_2| > 1$,
- soit $\widetilde{V_1} = V_1 \cap N(V_2)$ (les sommets de V_1 voisins de ceux de V_2) et $\widetilde{V_2} = V_2 \cap N(V_1)$, alors pour tout $u \in \widetilde{V_1}$ et tout $v \in \widetilde{V_2}$, $\{u, v\} \in E$.

Voir aussi Figure 5.10. Cette notion est intéressante car elle généralise celle des modules (un module $\mathcal{M}$ est un *split* où $V_1 \setminus \widetilde{V_1} = \emptyset$ et $\mathcal{M} = \widetilde{V_1}$).

On décompose récursivement un graphe en trouvant, s'il existe, un *split* $\{V_1, V_2\}$ et en itérant sur les deux graphes $G[V_1 \cup \{\widetilde{v_2}\}]$ et $G[V_2 \cup \{\widetilde{v_1}\}]$, avec $\widetilde{v_1} \in \widetilde{V_1}$ et $\widetilde{v_2} \in \widetilde{V_2}$. Une telle décomposition est également possible en temps linéaire [Dah00]. Peut-on l'utiliser pour généraliser le problème MODULE GRAPH MOTIF ?

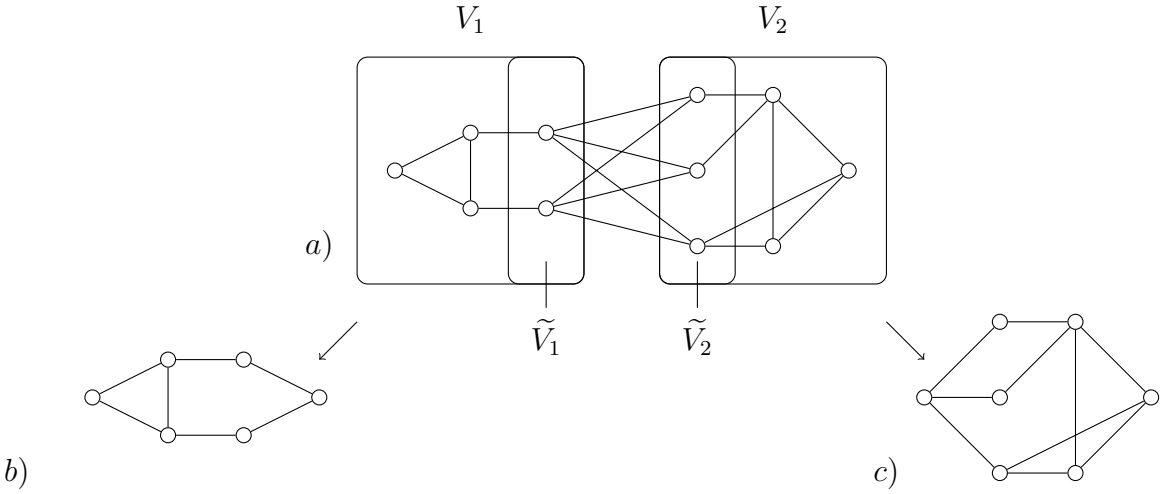


FIGURE 5.10 – Un exemple de *split* $\{V_1, V_2\}$ pour le graphe en $a)$. La décomposition est récursive sur les deux graphes en $b)$ et $c)$. Exemple tiré de [Pau06].

Questions algorithmiques

D'un point de vue algorithmique, il serait intéressant de savoir si le problème MODULE GRAPH MOTIF est W[1]-Difficile si le paramètre est le nombre de couleurs du motif, tout comme le problème EXACT GRAPH MOTIF, ou bien si l'utilisation de modules permet de changer de classe de complexité.

On remarque également qu'il est certainement possible d'obtenir un algorithme de complexité paramétrée pour le problème de comptage en adaptant l'algorithme de la Proposition 5.3.3.

Chapitre 6

Implémentations du problème GRAPH MOTIF et évaluations

Contenu

Nous avons vu dans les deux chapitres précédents que les variantes du problème GRAPH MOTIF sont NP-Difficiles même avec de fortes contraintes sur les entrées, mais également difficiles à approximer. De plus, certains algorithmes basés sur la technique du color-coding se prêtent difficilement à l'implémentation. Cependant, rapprocher la théorie à la pratique est important pour les applications biologiques.

Une première étape dans l'élaboration de logiciels est Motus, un algorithme de *branch-and-bound* [LFS06, Lac07]. Il est testé pour des motifs de taille 3 ou 4, sur les réseaux métaboliques.

Une seconde étape est le serveur web Torque [BHK+09]. Écrit en Python, ce logiciel combine une approche par programmation dynamique utilisant la technique du color-coding avec de la programmation entière (selon la taille du motif). Les principales limitations de Torque se trouvent dans le fait que c'est un serveur web (par conséquent, il est compliqué de le connecter à d'autres services, les performances dépendent directement du serveur si le code source n'est pas disponible et il n'est pas possible d'effectuer des tests en série sans interface graphique), qu'il ne donne qu'une seule solution et non un ensemble de solutions possibles, et qu'il ne gère que des motifs colorfuls.

Nous présentons dans la suite de ce chapitre le logiciel GraMoFoNe, une alternative à ce serveur web, tentant d'éviter les limitations citées précédemment. Tout comme Torque, notre logiciel est prévu pour être utilisé avec des réseaux d'interactions entre protéines. Peu de changements seraient toutefois requis pour l'utiliser avec d'autres

types de réseaux.

Par la suite, nous évaluons en pratique GraMoFoNe sur des données réelles. Enfin, nous avons également implémenté un algorithme utilisant la technique de recherche de monômes multilinéaires. Nous effectuons une comparaison de son temps d'exécution avec Torque et GraMoFoNe.

6.1 GraMoFoNe, un greffon Cytoscape

6.1.1 Méthodes et implémentation

Nous avons choisi d'implémenter le problème GRAPH MOTIF WITH GAPS où, en plus, un ensemble de couleurs est associé à chaque sommet du graphe d'entrée. Ainsi, notre implémentation sera capable de prendre en compte la quasi-totalité des variantes présentées précédemment. On nomme *insertions* les couleurs présentes dans la solution mais qui n'étaient pas dans le motif et *délétions* les couleurs présentes dans le motif mais absentes de la solution. Pour plus de commodités, on considère par la suite que le nombre d'insertions (resp. délétions) est borné par un entier N_{ins} (resp. N_{del}) fixé.

Si les insertions et les délétions ne sont pas autorisées ($N_{ins} = N_{del} = 0$), on résout le problème LIST-COLORED GRAPH MOTIF. Si en plus on ne donne qu'une seule couleur par sommet du graphe, on résout le problème EXACT GRAPH MOTIF (pour des motifs colorful ou non). En autorisant uniquement des délétions, on résout le problème MAXIMUM GRAPH MOTIF.

En collaboration avec Guillaume Blin et Stéphane Vialette, nous avons développé un outil appelé GraMoFoNe (pour Graph Motif For Network), publié dans [BSV10a] et implémenté sous la forme d'un greffon (ou *plugin*) pour Cytoscape. Cytoscape [SMO+03] est une plateforme gratuite et open-source écrite en Java, maintenue à jour et largement utilisée par la communauté (des centaines d'articles l'utilisent) pour la visualisation et l'analyse de réseaux biologiques. Cytoscape permet également le développement d'outils extérieurs sous forme de greffon Java afin d'enrichir ses fonctionnalités. En 2011, plus d'une centaine de greffons sont disponibles, permettant, entre autres, d'effectuer certaines analyses, de permettre différentes visualisations de réseaux, de supporter d'autres formats de fichiers ou encore de se connecter à des bases de données. Proposer un greffon Cytoscape plutôt qu'un serveur web permet une meilleure vision de la part de la communauté, ainsi que l'utilisation des fonctionnalités propres à Cytoscape.

Notre greffon recherche les occurrences d'un motif défini par l'utilisateur dans un réseau chargé au préalable dans l'espace de travail de Cytoscape (plusieurs formats sont pris en charge). Pour effectuer cette tâche, nous utilisons un algorithme exact de résolution de programmation pseudo-booléenne (voir Section 1.3.6). Un grand nombre de solveurs de programation pseudo-booléenne existe. Nous avons décidé d'utiliser la librairie Java SAT4JPseudo [LBP07] pour trois raisons principales : (i) son intégration

aisée dans un environnement Java, (ii) ses bons résultats dans l'évaluation internationale Pseudo-Boolean 2007 [1] et (iii) sa gratuité (malheureusement, certains solveurs efficaces sont parfois chers).

On rappelle que l'on cherche comme solution un sous-ensemble V_T des sommets de G, tel que $G[V_T]$ soit connexe ($G[V_T]$ n'est pas forcement un arbre ici) respectant un multi-ensemble M, construit sur un ensemble de couleurs C. On dispose d'une fonction $col : V \rightarrow 2^C$ donnant l'ensemble des couleurs associé à chaque sommet du graphe. On note $occ_M(c)$ le nombre d'occurrences de la couleur $c \in C$ dans le motif M.

Comme précisé précédemment, nous autorisons les insertions et les délétions. Par conséquent, $|V_T|$ peut être différent de $|M|$. Plus précisément, si $|V_T| < |M|$ (resp. $|V_T| > |M|$), il y a au moins $|M| - |V_T|$ délétions (resp. $|V_T| - |M|$ insertions). Cependant, comparer $|M|$ et $|V_T|$ n'est pas suffisant pour déterminer le nombre exact d'insertions et de délétions dans la solution et ainsi respecter les bornes N_{del} et N_{ins}. En effet, s'il y a délétion d'une couleur et insertion d'une autre couleur, alors la taille de la solution est bien égale à la taille du motif (c'est-à-dire $|V_T| = |M|$), mais l'ensemble des couleurs de la solution ne respecte pas le motif (c'est-à-dire $col(V_T) \neq M$). Pour faire face à ce problème, nous allons considérer pour chaque couleur c la différence entre le nombre d'occurrences de c dans le motif et le nombre de sommets colorés par c dans la solution, avant de faire la somme de ces différences pour obtenir le nombre total d'insertions et de délétions.

Un autre point délicat apparaît quand un sommet v de la solution contient plus d'une seule couleur. On cherche une bijection entre les couleurs de la solution et le motif, c'est-à-dire une bijection $f : V_T \rightarrow M$ telle que $\forall v \in V_T, f(v) \in col(v)$. Alors, un sommet v de la solution ne doit satisfaire qu'une seule couleur de M, disons c_1. Les couleurs de $col(v) \setminus \{c_1\}$ ne doivent pas être satisfaites dans M par le seul ajout de v dans la solution. Notre formulation gère également ce cas.

On présente maintenant la fonction objectif de notre formulation pseudo-booléenne, ainsi que les 23 contraintes définies sur 9 domaines de variables.

Variables. Pour chaque sommet $v \in V$, on définit une variable $x_v \in \{0, 1\}$ pour symboliser la présence de v dans la solution V_T : $x_v = 1$ si et seulement si $v \in V_T$. Pour chaque arête $\{u, v\} \in E$, on définit une variable $e_{uv} \in \{0, 1\}$ pour symboliser la présence de $\{u, v\}$ dans $G[V_T]$: $e_{uv} = 1$ si et seulement si $\{u, v\} \in G[V_T]$.

Comme expliqué plus loin, on définit $k + N_{ins}$ labels différents associés aux sommets de V_T pour assurer la connexité de $G[V_T]$. Un sommet porte un label seulement s'il appartient à la solution. Par conséquent, pour chaque sommet $v \in V$, on définit $k + N_{ins}$ variables $Label[v][i] \in \{0, 1\}$, avec $1 \leq i \leq k + N_{ins}$, pour représenter le *label* de v : $Label[v][i] = 1$ si et seulement si v porte le label i.

Pour chaque sommet $v \in V$ et chaque couleur $c \in col(v)$, on définit les variables $ColV[v][c]$ pour représenter la couleur de v dans la solution : $ColV[v][c] = 1$ si et seule-

1. http://www.cril.univ-artois.fr/PB07/

ment si on considère que v utilise la couleur c dans V_T. Ces variables sont utilisées pour déterminer la couleur choisie depuis les $|col(v)|$ couleurs de v. Comme nous l'avons déjà vu, en autorisant un ensemble de couleurs pour v, si v est dans la solution, ce sommet peut satisfaire jusqu'à $|col(v)|$ couleurs du motif. Comme on désire une bijection entre les couleurs de la solution et le motif, on doit choisir quelle est l'unique couleur considérée pour un sommet donné.

Pour chaque couleur $c \in C$, on définit $N_{ins} + 1$ variables $nins_c[i]$, $0 \leq i \leq N_{ins}$, pour représenter le nombre d'insertions pour la couleur c : $nins_c[i] = 1$ si et seulement s'il y a i insertions de sommets portant la couleur c. De manière équivalente, pour chaque couleur $c \in C$, on définit $N_{del} + 1$ variables $ndel_c[i]$, $0 \leq i \leq N_{del}$, pour représenter le nombre de délétions pour la couleur c : $ndel_c[i] = 1$ si et seulement si il y a i délétions pour la couleur c.

Pour chaque couleur $c \in C$, on définit trois variables $\mathrm{IsExact}_c$, IsIns_c et IsDel_c pour indiquer s'il y a des insertions ou des délétions pour c dans la solution : $\mathrm{IsExact}_c = 1$ (resp. $\mathrm{IsIns}_c = 1$, $\mathrm{IsDel}_c = 1$) si et seulement si le nombre de sommets dans V_T avec la couleur c est égal à (resp. plus grand que , plus petit que) $occ_M(c)$. Ces variables sont utilisées uniquement pour faciliter la lecture – il y a en effet équivalence entre $nins_c[0]$ et $ndel_c[0]$, et ces variables.

Objectif. L'objectif de notre programme pseudo-booléen est de maximiser le score de la solution. On cherche à maximiser la somme de toutes les variables e_{uv}. Formellement, l'objectif est : $\max \sum_{\{u,v\} \in E} e_{uv}$.

On remarque que l'on peut très facilement gérer une fonction de poids sur les arêtes du graphe G, $w : E \to \mathbb{R}$. En supposant que w donne la probabilité que l'interaction existe réellement, on cherche à maximiser la somme des poids des arêtes de la solution. Alors, la fonction objectif est $\max \sum_{\{u,v\} \in E} e_{uv} w(\{u, v\})$.

Contraintes. On définit maintenant l'ensemble des contraintes utilisées dans notre formulation pseudo-booléenne.

Les deux contraintes suivantes contrôlent que la solution V_T soit de taille correcte (selon $|M|$, N_{ins} et N_{del}).

$$\forall u, v \in V, \quad e_{uv} \Leftrightarrow x_u \wedge x_v, \tag{6.1}$$

$$|M| - N_{del} \leq \sum_{v \in V} x_v \leq |M| + N_{ins}. \tag{6.2}$$

La contrainte (6.1) assure que $\{u, v\} \in G[V_T]$ si et seulement si $u \in V_T$ et $v \in V_T$. La contrainte (6.2) contrôle le nombre de sommets dans la solution. Si les insertions et les délétions sont interdites, la taille de la solution doit être égale à $|M|$. Si des insertions (resp. délétions) sont autorisées, la taille de la solution peut alors être plus grande (resp. plus petite) que $|M|$.

Les quatre contraintes suivantes assurent la connexité de $G[V_T]$.

$$\forall v \in V, \quad x_v \Rightarrow \left(\sum_{i=1}^{|M|+N_{ins}} Label[v][i] = 1 \right), \tag{6.3}$$

$$\forall v \in V, \quad \neg x_v \Rightarrow \left(\sum_{i=1}^{|M|+N_{ins}} Label[v][i] = 0 \right), \tag{6.4}$$

$$\forall 1 \leq i \leq |M| + N_{ins}, \quad \sum_{v \in V} Label[v][i] \leq 1, \tag{6.5}$$

$$\forall v \in V, \forall 1 \leq i < |M| + N_{ins}, Label[v][i] \Rightarrow \left(\sum_{u \in N(v)} \sum_{j>i} Label[u][j] \geq 1 \right). \tag{6.6}$$

La contrainte (6.3) assure que si $v \in V_T$, alors v a exactement un label, qui est un entier entre 1 et $|M| + N_{ins}$. La contrainte (6.4) assure que si $v \notin V_T$, alors v n'a pas de label. La contrainte (6.5) assure que chaque label est attribué à au plus un sommet – en effet, à cause des délétions, certains labels peuvent ne pas être attribués. Enfin, la contrainte (6.6) assure la connexité de $G[V_T]$: chaque sommet de la solution V_T (sauf celui avec le label maximum) doit avoir au moins un voisin dans $G[V_T]$ avec un label plus grand que le sien (voir aussi Figure 6.1).

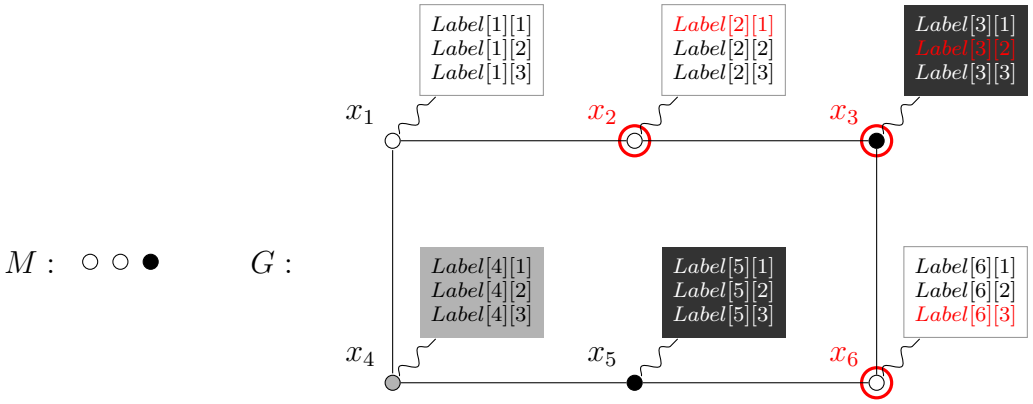


FIGURE 6.1 – Exemple où les sommets de G sont colorés par trois couleurs différentes (blanc, noir et gris), le motif demande deux occurrences de blanc et une occurrence de noir, et $N_{ins} = N_{del} = 0$. Pour chaque sommet, $|M| + N_{ins} = 3$ labels sont donnés. Les sommets 2, 3 et 6 forment une solution possible pour le motif M (en rouge sur la figure). Alors, si $x_2 = x_3 = x_6 = 1$ et $x_1 = x_4 = x_5 = 0$, la contrainte (6.2) est bien respectée puisque la somme des $x_i, 1 \leq i \leq 6$, est égale à 3. Les contraintes sur les couleurs sont également respectées, par exemple la somme des variables dont le sommet porte la couleur blanche ($x_1 + x_2 + x_6$) est égale à $occ_M(blanc) = 2$ (contrainte (6.7)). Enfin, si on donne le label 1 au sommet 2, le label 2 au sommet 3 et le label 3 au sommet 6, on voit que chaque sommet de la solution (sauf celui avec le label le plus haut) possède un voisin avec un label supérieur – d'où la connexité.

Les deux contraintes suivantes assurent que $G[V_T]$ contienne le bon nombre de sommets colorés par c, pour chaque $c \in C$, selon $occ_m(c)$, N_{ins} et N_{del}.

$$\forall c \in C, occ_M(c) - N_{del} \leq \sum_{\substack{v \in V \\ c \in col(v)}} x_v \leq occ_M(c) + N_{ins}, \tag{6.7}$$

$$\forall v \in V, \quad \sum_{c \in col(v)} ColV[v][c] = x_v. \tag{6.8}$$

La contrainte (6.7) assure un nombre suffisant de sommets colorés par c dans V_T pour chaque couleur $c \in C$. Si les insertions et les délétions sont interdites, alors une solution doit contenir exactement $occ_M(c)$ occurrences de c, pour chaque $c \in C$. Quand des insertions (resp. délétions) sont autorisées, une couleur peut apparaître plus souvent (resp. moins souvent). La contrainte (6.8) assure que pour chaque sommet $v \in V_T$, une unique couleur est choisie parmi les $|col(v)|$ couleurs associées à v.

Les trois contraintes suivantes assurent que soit toutes les occurrences d'une couleur $c \in C$ apparaissent dans la solution, soit au moins une de ces occurrences a subi une délétion, soit il y a au moins une occurrence de c qui a été insérée.

$$\forall c \in C, \quad \mathrm{IsExact}_c + \mathrm{IsIns}_c + \mathrm{IsDel}_c = 1, \tag{6.9}$$

$$\forall c \in C, \sum_{v \in V} ColV[v][c] - occ_M(c) \leq \mathrm{IsIns}_c \,.N_{ins} - \mathrm{IsDel}_c, \tag{6.10}$$

$$\forall c \in \mathcal{C}, \quad \sum_{v \in V} ColV[v][c] - occ_M(c) \geq \neg\, \mathrm{IsExact}_c - \mathrm{IsDel}_c - \mathrm{IsDel}_c \,.N_{del}. \tag{6.11}$$

Pour chaque couleur, la contrainte (6.9) assure la cohérence entre les trois variables $\mathrm{IsExact}_c$, IsIns_c et IsDel_c, tandis que les contraintes (6.10) et (6.11) assurent la cohérence entre $ColV, \mathrm{IsExact}, \mathrm{IsIns}$ et IsDel : $\forall c \in C$, $\mathrm{IsExact}_c$ (resp. IsIns_c, IsDel_c) = 1 si et seulement si $\sum_{v \in V} ColV[v][c] - occ_M(c) = 0$ (resp. > 0, < 0).

Les six contraintes suivantes assurent que le nombre d'insertions est inférieur à N_{ins}.

$$\forall c \in C, \quad \sum_{i=0}^{N_{ins}} nins_c[i] = 1, \tag{6.12}$$

$$\forall c \in C, \quad \mathrm{IsIns}_c \Rightarrow nins_c[0] = 0, \tag{6.13}$$

$$\forall c \in C, \quad \neg\, \mathrm{IsDel}_c + nins_c[0] \geq 1, \tag{6.14}$$

$$\forall c \in C, \forall 0 \leq i \leq N_{ins}, \sum_{v \in V} ColV[v][c] - occ_M(c) \leq i.nins_c[i] + \neg nins_c[i].N_{ins}, \tag{6.15}$$

$$\forall c \in C, \forall 0 \leq i \leq N_{ins},$$
$$\neg nins_c[i] + \sum_{v \in V} ColV[v][c] - occ_M(c) + N_{del}.\,\mathrm{IsDel}_c \geq i.nins_c[i], \tag{6.16}$$

$$\sum_{c \in C} \sum_{i=1}^{N_{ins}} i.nins_c[i] + \sum_{\substack{v \in V \\ col(v)=\emptyset}} x_v \leq N_{ins}. \tag{6.17}$$

La contrainte (6.12) assure qu'étant donnée une couleur $c \in C$, il existe un unique i tel que $nins_c[i] = 1$. La contrainte (6.13) assure que les variables $nins_c$ et IsIns_c sont cohérentes. La contrainte (6.14) assure qu'étant donnée une couleur $c \in C$, il y a soit des insertions, soit des délétions. Les contraintes (6.15) et (6.16) assurent que $nins_c[i] = 1$ si et seulement si il y a i insertions de la couleur $c \in C$ (c'est-à-dire si la différence entre $\sum_{v \in V} ColV[v][c]$ et $occ_M(c)$ est égale à i). La contrainte (6.17) assure que le nombre d'insertions est borné par N_{ins}. La somme de toutes les insertions pour une couleur donnée ainsi que les insertions de sommets non colorés doit être inférieur à N_{ins}.

On donne six contraintes équivalentes aux six précédentes afin de contrôler que le nombre de délétions est inférieur à N_{del}.

$$\forall c \in C, \quad \sum_{i=0}^{N_{del}} ndel_c[i] = 1, \tag{6.18}$$

$$\forall c \in C, \quad \mathrm{IsDel}_c \Rightarrow ndel_c[0] = 0, \tag{6.19}$$

$$\forall c \in C, \quad \neg\,\mathrm{IsIns}_c + ndel_c[0] \geq 1, \tag{6.20}$$

$$\forall c \in C, \forall 0 \leq i \leq N_{del}, \; -\sum_{v \in V} ColV[v][c] + occ_M(c) \leq i.ndel_c[i] + \neg ndel_c[i].N_{del}, \tag{6.21}$$

$$\forall c \in C, \forall 0 \leq i \leq N_{del}, \neg ndel_c[i] - \sum_{v \in V} ColV[v][c] + occ_M(c) + N_{ins}.\,\mathrm{IsIns}_c \geq i.ndel_c[i], \tag{6.22}$$

$$\sum_{c \in C} \sum_{i=1}^{N_{del}} i.ndel_c[i] \leq N_{del}. \tag{6.23}$$

La contrainte (6.18) assure qu'étant donnée une couleur $c \in C$, il existe un unique i tel que $ndel_c[i] = 1$. La contrainte (6.19) assure que les variables $ndel_c$ et IsDel_c sont cohérentes. La contrainte (6.20) assure que pour une couleur $c \in C$, il y a soit des

délétions, soit des insertions. Les contraintes (6.21) et (6.22) assurent que $ndel_c[i] = 1$ si et seulement si il y a i délétions pour la couleur $c \in C$ (c'est-à-dire la différence entre $occ_M(c)$ et $\sum_{v \in V} ColV[v][c]$ est égale à i). Enfin, la contrainte (6.23) assure que le nombre de délétions est borné par N_{del}. La somme de toutes les délétions pour une couleur donnée doit être inférieure à N_{del}.

Prouvons maintenant que l'ensemble de ces contraintes permet la résolution du problème.

Proposition 6.1.1. *La formulation pseudo-booléenne donnée précédemment permet de résoudre le problème* GRAPH MOTIF WITH GAPS*, où un ensemble de couleur est associé à chaque sommet.*

Démonstration. On prouve dans un premier temps que la proposition est vraie si les insertions et les délétions ne sont pas autorisées.

Prouvons d'abord qu'une solution au problème peut être trouvée par notre formulation pseudo-booléenne, c'est-à-dire que depuis la solution au problème, on peut donner une assignation des variables respectant l'ensemble des contraintes.

Étant donnée une solution $T = (V_T, E_T)$ au problème, on place $x_v = 1$ si $v \in V_T$; $x_v = 0$ sinon, et $e_{u,v} = 1$ si $u \in V_T$ et $v \in V_T$, $e_{u,v} = 0$ sinon. Comme T est un arbre, on donne un label aux sommets de V_T selon un parcours postfixe de T.

Comme les insertions et les délétions sont interdites, $|V_T| = |M|$. Par définition, exactement $|M|$ variables x_v sont égales à 1, par conséquent, les contraintes (6.1) et (6.2) sont vraies. Le parcours postfixe de T donnant les labels assure que toutes les variables x_v telles que $v \in V_T$, ont un label unique et distinct. Par conséquent, les contraintes (6.3) et (6.5) sont vraies. Comme aucun label n'est donné aux variables $x_v, v \notin V_T$, la contrainte (6.4) est vraie. De plus, selon le parcours postfixe de T, le père de chaque sommet v, sauf la racine, possède un label plus grand que v. Par conséquent, dans T, chaque sommet (sauf la racine) possède au moins un voisin avec un label plus grand que le sien. Donc, la contrainte (6.6) est vraie. Étant donnée la bijection $f : V_T \to M$, il existe exactement $occ_M(c)$ occurrences de chaque couleur $c \in C$ dans les couleurs de V_T et donc, la contrainte (6.7) est vraie. De plus, il y a seulement une image $f(v)$ associée à chaque $v \in V_T$, donc la somme de la contrainte (6.8) est égale à 1, et la contrainte est vraie si $x_v = 1$. Par cette même bijection f, chaque élément de M est associé à un élément de V_T. Par conséquent, il n'existe pas de sommet $v \notin V_T$ ayant une image dans M par f, la somme de la contrainte (6.8) est égale à 0 et la contrainte est également vraie si $x_v = 0$. Comme ni les insertions, ni les délétions ne sont autorisées, chaque couleur c apparait exactement $occ_M(c)$ fois dans la solution. Donc, la contrainte (6.9) est vraie. De plus, $\sum_{v \in V} ColV[v][c] - occ_M(c)$ est égal à 0. Donc les contraintes (6.10) et (6.11) sont vraies si et seulement si $\mathrm{IsExact}_c = 1$. En effet, si $\mathrm{IsIns}_c = 1$, la contrainte (6.11) n'est pas vraie ($0 \geq 1$), et si $\mathrm{IsDel}_c = 1$, la contrainte (6.10) n'est pas vraie ($0 \leq -1$). Pour chaque couleur $c \in C$, les contraintes (6.12) à (6.23) sont vraies si $nins_c[0] = 1$ et $ndel_c[0] = 1$, ce qui est le cas car les insertions et les délétions sont interdites.

Prouvons maintenant qu'étant donnée une solution à notre programme pseudo-booléen, on trouve une solution pour le problème GRAPH MOTIF WITH GAPS où un ensemble de couleurs est associé à chaque sommet du graphe.

Étant donnée une solution au programme pseudo-booléen, on construit $V_T = \{v : x_v = 1\}$. Selon les contraintes (6.2) et (6.7), V_T et M sont des ensembles finis de même taille, $|V_T| = |M|$. On construit $f : V_T \to M$ de la manière suivante. Pour chaque v tel que $x_v = 1, f(v) = c$, où c est tel que $ColV[v][c] = 1$. Par la contrainte (6.8), il existe exactement un $c \in col(v)$ tel que $ColV[v][c] = 1$ si $x_v = 1$. Par conséquent, f est une bijection. Il reste à montrer que $G[V_T]$ est connexe.

Par la contrainte (6.3), chaque sommet $v \in V_T$ possède un label. Soit r le sommet avec le plus grand label. La contrainte (6.5) assure que ce label est unique. Montrons qu'il existe un chemin dans V_T reliant chaque sommet $v \in V_T$ à r. Pour cela, prouvons par induction qu'il existe un chemin de v à r avec des labels croissants, $\forall v \in V_T$. Le cas $v = r$ est trivial. Supposons qu'il existe un chemin p dans V_T de longueur $|p|$, depuis v avec des labels croissants. Soit p_l le dernier sommet de p. Si $p_l = r$, la propriété est satisfaite. Sinon, par la contrainte (6.6), p_l possède au moins un voisin u avec un label plus grand que le sien. Alors, il existe un chemin $p \cup \{u\}$ de longueur $|p| + 1$ avec des labels croissants.

Enfin, comme $N_{ins} = N_{del} = 0$, les contraintes (6.10) et (6.11) impliquent que pour chaque $c \in C$, $\neg\,\mathrm{IsExact}_c - \mathrm{IsDel}_c \leq \sum_{v \in V} ColV[v][c] - occ_M(c) \leq -\mathrm{IsDel}_c$. Comme la contrainte (6.9) est vraie, $IsExact_c = 1$ et $IsIns_c = IsDel_c = 0$, et donc pour chaque $c \in C$, la différence entre le nombre de sommets colorés par c et le nombre d'occurrences de c dans le motif est exactement égale à 0.

Prouvons maintenant la Proposition 6.1.1 quand les insertions et les délétions sont autorisées. On montre dans un premier temps que les contraintes (6.9) à (6.11) sont cohérentes quand les insertions et les délétions sont autorisées. Nous avons déjà montré qu'elle étaient vraies si les insertions et les délétions étaient interdites.

- S'il existe i insertions pour une couleur c, alors $\sum_{v \in V} ColV[v][c] - occ_M(c) = i$. La contrainte (6.9) assure que seule une variable parmi $\mathrm{IsExact}_c, \mathrm{IsIns}_c$ et IsDel_c est égale à 1.
  - Si $\mathrm{IsExact}_c = 1$, alors les contraintes (6.10) ($i \leq 0$) et (6.11) ($i \geq 0$) sont toutes les deux vraies si et seulement si $i = 0$.
  - Si $\mathrm{IsIns}_c = 1$, alors les contraintes (6.10) ($i \leq N_{ins}$) et (6.11) ($i \geq 0$) sont toutes les deux vraies si et seulement si $0 \leq i \leq N_{ins}$, ce qui est notre supposition.
  - Si $\mathrm{IsDel}_c = 1$, alors la contrainte (6.10) ($i \leq -1$) est fausse puisque le nombre d'insertions est positif ($i > 0$).
- S'il existe d délétions, le raisonnement est identique.

Montrons maintenant que les contraintes (6.12) à (6.16) et les contraintes (6.18) à (6.22) sont cohérentes avec le nombre d'insertions et délétions pour une couleur donnée dans une solution au problème.

– S'il existe i insertions pour une couleur c, alors $\sum_{v \in V} ColV[v][c] - occ_M(c) = i$ et $\mathrm{IsIns}_c = 1, \mathrm{IsDel}_c = \mathrm{IsExact}_c = 0$. Les contraintes (6.12) et (6.13) assurent qu'il existe un $i \neq 0$ tel que $nins_c[i] = 1$. La contrainte (6.14) est vraie car $\mathrm{IsDel}_c = 0$. Les contraintes (6.15) et (6.16) sont toutes les deux vraies si et seulement si $nins_c[i] = 1$ ($i \leq i$ et $i \geq i$). Sinon, si $nins_c[j] = 1, j \neq i$, les contraintes (6.15) et (6.16) sont vraies si et seulement si $i \leq j$ et $i \geq j$, ce qui est impossible car $j \neq i$.
  Comme $\mathrm{IsIns}_c = 1$, la contrainte (6.20) est vraie si et seulement si $ndel_c[0] = 1$. Alors, la contrainte (6.18) est vraie. Comme $\mathrm{IsDel}_c = 0$, alors la contrainte (6.19) est vraie. Ensuite, les contraintes (6.21) ($-i \leq 0$) et (6.22) ($-i + N_{ins} \geq 0$) sont vraies quand $ndel_c[0] = 1$.
– S'il existe d délétions pour une couleur c, le raisonnement est similaire.

Dans les deux cas, les contraintes (6.17) et (6.23) sont vraies si et seulement si le nombre total d'insertions et de délétions sont respectivement bornées par N_{ins} et N_{del}.

Inversement, étant donnée une solution de notre formulation pseudo-booléenne, prouvons que le nombre d'insertions et de délétions dans la solution construite est cohérent.

Par la contrainte (6.9), pour chaque couleur $c \in C$, seuls trois cas sont possibles si toutes les contraintes sont respectées. Nous avons déjà vu le cas où $\mathrm{IsExact}_c = 1$.

Observons donc le cas où $\mathrm{IsIns}_c = 1$ (et donc $\mathrm{IsExact}_c = \mathrm{IsDel}_c = 0$). Comme les contraintes (6.10) et (6.11) sont vraies, on sait que pour chaque $c \in C$, $0 \leq \sum_{v \in V} ColV[v][c] - occ_M(c) \leq N_{ins} - 0$. Il y a donc au plus N_{ins} occurrences de c de plus dans la solution que dans le motif. Par les contraintes (6.12) et (6.13), il existe un seul $1 \leq i \leq N_{ins}$ tel que $nins_c[i] = 1$. Pour ce i, par les contraintes (6.15) et (6.16), on sait que $i \leq \sum_{v \in V} ColV[v][c] - occ_M(c) \leq i$, donc que la différence entre les sommets colorés par c dans la solution et le nombre d'occurrences de c dans le motif est exactement égale à i. Enfin, la contrainte (6.17) indique que la somme des i tels que $nins_c[i] = 1$ plus la somme des $x_v = 1$ tels que v n'a pas de couleur est inférieur à N_{ins} – il y a donc au plus N_{ins} insertions dans la solution construite.

La preuve quand $\mathrm{IsDel}_c = 1$ (et donc $\mathrm{IsExact}_c = \mathrm{IsIns}_c = 0$) est similaire. □

On décrit maintenant deux étapes de pré-traitement pour accélérer la résolution du programme pseudo-booléen.

Premièrement, on remarque qu'une protéine du motif n'ayant aucun homologue dans le réseau (c'est-à-dire une couleur présente dans le motif mais qui n'apparaît pas dans le graphe) sera considérée comme une délétion dans n'importe quelle solution. Soit $D \subseteq M$ l'ensemble de couleurs de M n'apparaissant pas dans G. Si $|D| > N_{del}$, alors aucune solution n'est possible pour ce motif. Sinon, les couleurs de D sont obligatoirement des délétions dans n'importe quelle solution. Par conséquent, le programme pseudo-booléen peut être lancé avec le motif $M \setminus D$ et $N_{del} - |D|$ délétions autorisées.

Ensuite, comme montré dans [BHK+09], il est possible d'élaguer le graphe en supprimant certains sommets. Un sommet de G sans aucune couleur lui étant associé peut

être trop éloigné d'un sommet coloré (relativement à la longueur du plus court chemin) pour être inséré dans une solution, selon le nombre d'insertions autorisé au maximum. Plus précisément, un sommet non coloré $u \in V$ est gardé dans G uniquement s'il existe deux sommets colorés v_1 et v_2 tels que $dist(u, v_1) + dist(u, v_2) \leq N_{ins} + 1$, où $dist(u, v)$ est la longueur du plus court chemin entre u et v. Si cette contrainte n'est pas respectée, u ne peut être dans aucune solution et peut donc être supprimé de G.

Une fois G élagué, il est possible qu'il contienne plusieurs composantes connexes. Par définition, une solution (connexe) ne peut se trouver que dans une unique composante connexe. Le programme pseudo-booléen est donc lancé sur chaque composante connexe *valide*. Une composante connexe est dite valide si elle contient au moins $|M| - N_{del}$ couleurs différentes de M. Dans le cas contraire, une solution ne pourrait jamais être présente dans une telle composante connexe et donc, il n'est pas nécessaire de la considérer. Ces différentes étapes de pré-traitement peuvent s'avérer en pratique très performantes. En effet, selon [BHK+09], en pratique, seul 5% des sommets sont colorés dans G. Cependant, ces réseaux sont très connexes. Par conséquent, toujours selon [BHK+09], en autorisant plus de 3 insertions, 99% du réseau est gardé.

6.1.2 Fonctionnalités

Une capture d'écran du greffon GraMoFoNe se trouve Figure 6.2 (d'autres sont disponibles sur le site web dédié à GraMoFoNe [1]). L'utilisateur fournit les données d'entrées ainsi que les paramètres sur la partie gauche de l'espace de travail. Le réseau biologique est dessiné au centre de l'espace de travail tandis que les résultats sont présentés sur la droite. Décrivons maintenant les entrées et sorties de GraMoFoNe.

Entrées

Le réseau et le motif. Le réseau doit être chargé dans l'environnement Cytoscape. Le motif peut être (i) un motif prédéfini fourni avec le greffon, (ii) entré manuellement par l'utilisateur dans la boîte de dialogue appropriée ou (iii) chargé depuis un fichier FASTA.

Homologies. Nous considérons deux protéines homologues (et donc portant la même couleur) selon une analyse de leur séquence *via* BLASTp, logiciel effectuant une telle tâche. Ce dernier nécessite des fichiers FASTA contenant les séquences de chaque protéine utilisée – c'est pourquoi on demande à l'utilisateur de donner les fichiers FASTA correspondant au motif ainsi qu'au réseau. Si l'utilisateur ne possède pas de tels fichiers, le greffon essaie de les récupérer depuis la base de données Uniprot [The11], *via* le service web EBI [MVG+09].

L'utilisateur peut également spécifier le seuil à partir duquel BLASTp considère deux protéines comme étant homologues.

Insertions et délétions. L'utilisateur peut fournir le nombre maximum de délétions

1. http://igm.univ-mlv.fr/AlgoB/gramofone/

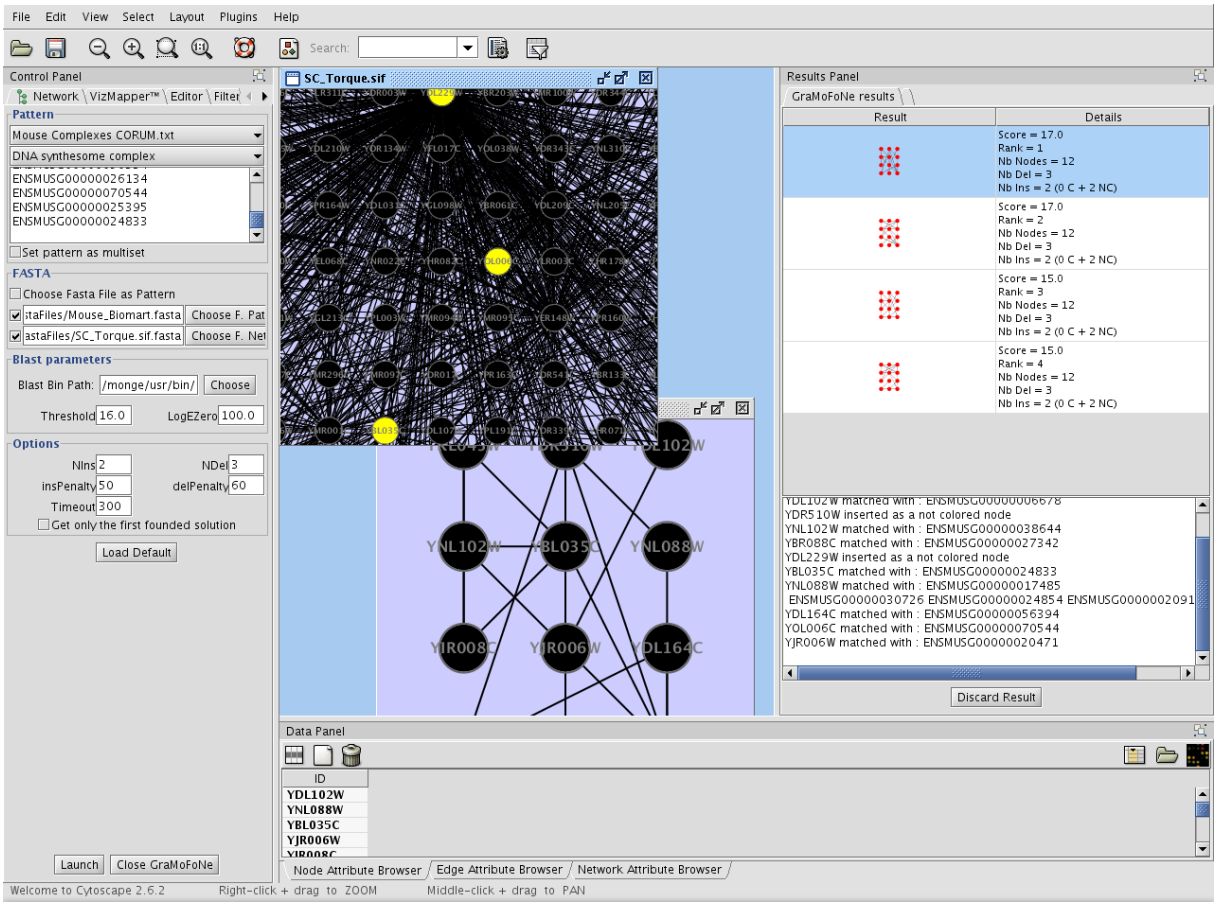


FIGURE 6.2 – Capture d'écran de GraMoFoNe, greffon chargé dans Cytoscape. Sur la gauche, le motif utilisé est prédéfini parmi ceux de la souris (DNA synthesome complex). On trouve au centre le réseau (de la levure) dans lequel est cherché le motif, avec en jaune un extrait de la solution. En arrière plan, la solution extraite en tant que nouveau réseau. À droite, quatre solutions possibles.

et d'insertions autorisées dans une solution, ainsi que la valeur de la pénalité associée dans le score final d'une solution.

Sorties

Une fois la recherche de solution lancée, le greffon fournit la liste des sous-réseaux de la solution, ordonnée par leurs scores (on rappelle que Torque ne fournit que la meilleure solution).

L'utilisateur peut ensuite mettre en évidence chaque solution dans le réseau complet, mais aussi l'exporter en tant que nouveau réseau. Le greffon donne également une correspondance possible entre le résultat et le motif.

À l'heure où nous écrivons ces lignes, notre greffon a été téléchargé plus de 350 fois *via* le service dédié de Cytoscape.

6.2 Évaluation pratique et comparaisons

Dans cette section, on cherche à valider GraMoFoNe sur des données réelles, en vérifiant que le greffon est capable de retrouver des complexes protéiques connus à travers plusieurs espèces. Pour cela, on recherche des complexes protéiques (motifs) de six espèces différentes dans trois réseaux d'interactions entre protéines grâce à un mode *batch* de notre greffon, ne nécessitant pas Cytoscape. De plus, on compare les temps d'exécution entre GraMoFoNe, Torque et une implémentation en Java d'un algorithme basé sur le comptage de monômes multilinéaires (Proposition 4.5.2), nommé par la suite MLD.

6.2.1 Acquisition des données

Les réseaux des espèces *Saccharomyces cerevisiae* (levure), *Drosophila melanogaster* (mouche) et *Homo sapiens* sont ceux fournis par les auteurs de Torque [BHK+09]. Ils ont été obtenus depuis des articles récents et des bases de données publiques. Ils contiennent entre 5 000 et 8 000 protéines, ainsi qu'entre 20 000 et 40 000 interactions. Les motifs sont des complexes protéiques de *Mus musculus* (souris), *Rattus norvegicus* (rat), *Bos taurus* (bovin), *Saccharomyces cerevisiae*, *Drosophila melanogaster* et *Homo sapiens*, également fournis par les auteurs de [BHK+09]. Les fichiers FASTA utilisés pour BLASTp ont été téléchargés depuis la base de données Biomart [SHB+09], avec l'ajout d'informations manuellement depuis les bases de données Uniprot [The11] et Ensembl [FAB+11].

6.2.2 Paramètres

Lors des tests, nous avons utilisé les mêmes paramètres entre les algorithmes. Nous avons en général choisi les mêmes que ceux présentés par les auteurs de Torque. Par conséquent, le seuil pour BLASTp est fixé à -$\log(10^{-7}) \simeq 16.1$. Lors de la recherche de motifs, deux insertions (N_{ins}) et délétions (N_{del}) sont autorisées pour les motifs de petite taille (inférieure à 7), trois pour les motifs de taille moyenne (taille entre 8 et 14) et quatre pour les motifs les plus grands. Cependant, de part la Proposition 4.5.2, MLD recherche uniquement des solutions exactement égales au motif. Par conséquent, lorsque l'on compare les temps de calculs entre les trois programmes, on fixe à 0 le nombre possible maximum d'insertions et délétions pour GraMoFoNe et Torque. Nous avons également fixé le temps limite pour la recherche d'une solution à 500 secondes.

6.2.3 Expérimentations

Nos tests sont effectués sur un PC standard de bureau (3GHz et 2Go de mémoire RAM). On rappelle que GraMoFoNe est basé sur la résolution d'un programme pseudo-booléen [BSV10a], que Torque est écrit en Python (nous n'utilisons pas la partie basée

sur la programmation entière) [BHK+09], tandis que l'implémentation de la Proposition 4.5.2 est faite en Java, avec l'aide de Khanh-Lam Mai [GS11].

On crée le motif y ajoutant une couleur distincte pour chaque protéine du complexe considéré. Ensuite, on colore le réseau de la manière suivante : un sommet représentant une protéine p_1 porte la couleur d'une protéine p_2 présente dans le motif si p_1 et p_2 sont homologues.

Le nombre de motifs retrouvé par Torque n'est pas recalculé, nous utilisons les valeurs données dans l'article [BHK+09].

Décrivons maintenant le calcul du nombre de motifs retrouvé par GraMoFoNe. On lance dans un premier temps les étapes de pré-traitement données précédemment. Ensuite, depuis la liste de motifs pour une espèce donnée, on garde uniquement les motifs *réalisables*. Un motif est dit réalisable si (i) sa taille est comprise entre 4 et 25, (ii) il y a moins de N_{del} couleurs du motif n'apparaissant pas dans G et (iii) il y a au moins une composante connexe avec suffisamment de couleurs pour retrouver le motif. Par la suite, pour un motif réalisable, le solveur indique qu'il a trouvé une solution (*Trouvé* sur la Figure 6.3), indique qu'il n'y a pas de solution pour ce motif dans ce réseau (*Non trouvé* sur la Figure 6.3), ou qu'il ne peut pas finir avant la limite de temps (*Timeout* sur la Figure 6.3).

Concernant MLD, l'algorithme ne supporte ni plusieurs couleurs sur les sommets du graphe, ni les insertions et délétions. Par conséquent, nous ne calculons pas le nombre de solutions trouvées par cet algorithme pour chaque motif, puisque les données biologiques sont trop imprécises pour que les résultats soient réalistes. Nous calculons seulement, pour chaque motif réalisable (avec $N_{ins} = N_{del} = 0$), le temps de calcul de l'algorithme pour trouver une solution, ou indiquer qu'il n'y a pas de solution.

6.2.4 Comparaison avec les travaux relatifs

6.2.4.1 Résultats biologiques

Nous avons lancé GraMoFoNe pour chaque motif réalisable de chaque espèce, avec le réseau de chaque espèce (sauf celui de l'espèce du motif). La comparaison entre le nombre de motifs retrouvé par GraMoFoNe et Torque se trouve Figure 6.3. Pour la plupart des espèces, GraMoFoNe trouve légèrement plus de motifs réalisables (c'est-à-dire la hauteur totale de chaque barre) et souvent légèrement plus de résultats positifs (c'est-à-dire la hauteur des barres blanches) que Torque. Ceci peut s'expliquer par les approches choisies lors des étapes de pré-traitement, ou à des différences au niveau des informations dans les fichiers FASTA.

Comme Torque, GraMoFoNe est capable de rechercher des motifs où aucune information n'est connue sur la topologie de ceux-ci. Toujours comme Torque, on remarque un plus grand nombre de résultat négatifs quand le motif est recherché dans le réseau de la mouche. Selon les auteurs de [BHK+09], ceci est dû au fait que les données concernant

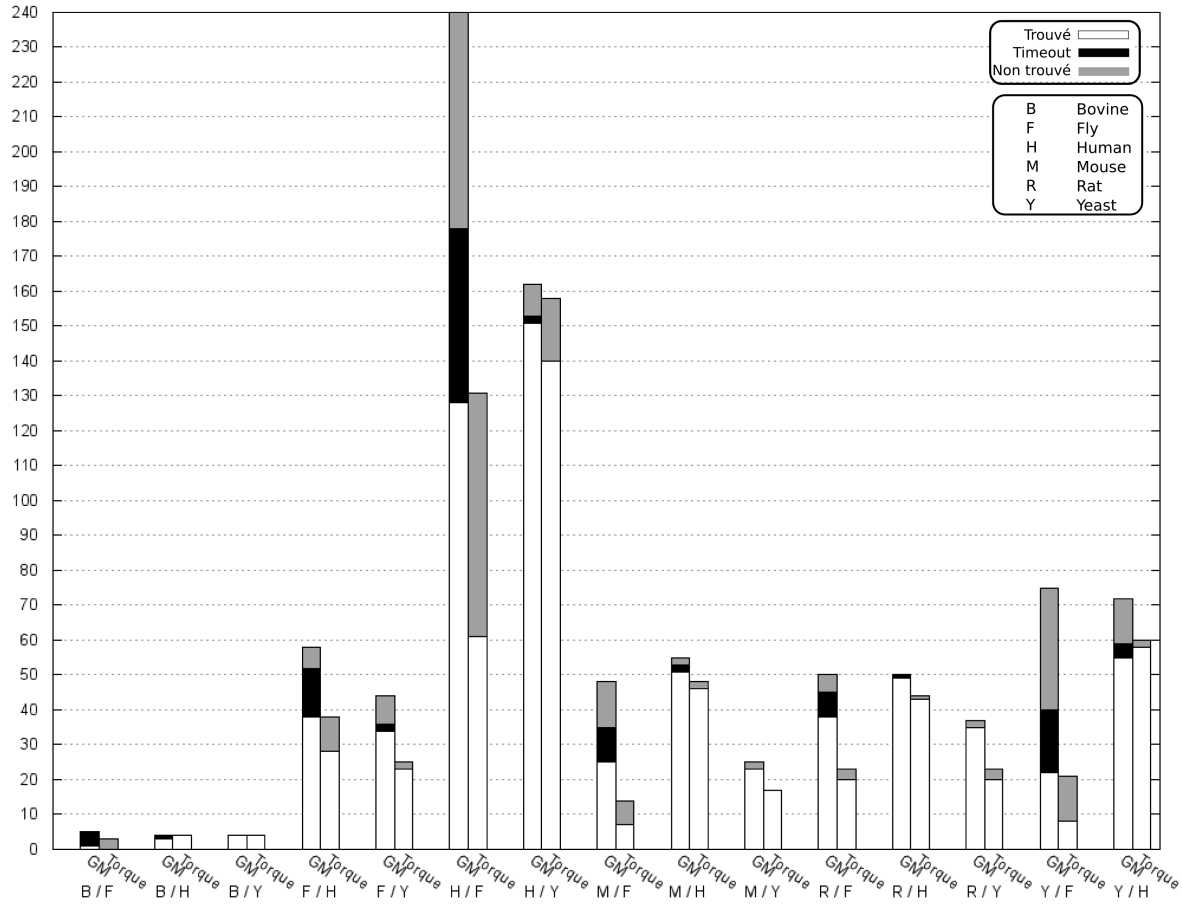


FIGURE 6.3 – Comparaison des résultats de recherche de motifs entre GraMoFoNe (GM) et Torque [BHK+09]. Chaque histogramme étiqueté par X/Y correspond à la recherche de l'ensemble des motifs de l'espèce X dans le réseau de l'espèce Y. Les barres blanches (resp. grises) correspondent aux motifs réalisables trouvés (resp. non trouvés) dans le réseau. Les barres noires correspondent aux motifs où le temps limite de calcul a été atteint avant d'obtenir un résultat. Par conséquent, la barre entière correspond au nombre de motifs réalisables. On précise que les valeurs pour Torque n'ont pas été recalculées – elles correspondent à celles indiquées dans l'article [BHK+09].

cette espèce sont de mauvaise qualité, avec notamment un important nombre de faux négatifs, c'est-à-dire des interactions manquantes. Un motif peut ne pas être trouvé si un faux négatif déconnecte une solution potentielle. Inversement, les faux positifs ne perturbent pas la connexité mais créent des mauvaises solutions.

On note qu'avec l'ensemble de paramètres choisis, il n'y a pas de différence significative en terme de nombre de solutions positives en considérant un motif comme multi-ensemble (où deux protéines homologues du complexe protéique possèdent la même couleur) ou comme un ensemble simple.

6.2.4.2 Temps de calcul

Le temps moyen de calcul de GraMoFoNe, Torque et de MLD se trouve Figure 6.4. Pour obtenir une comparaison valide, ni les insertions ni les délétions n'étaient autorisées pour GraMoFoNe et Torque. Par conséquent, le nombre de motifs réalisables dans ces très fortes conditions est très faible pour des motifs de taille plus grande que 9.

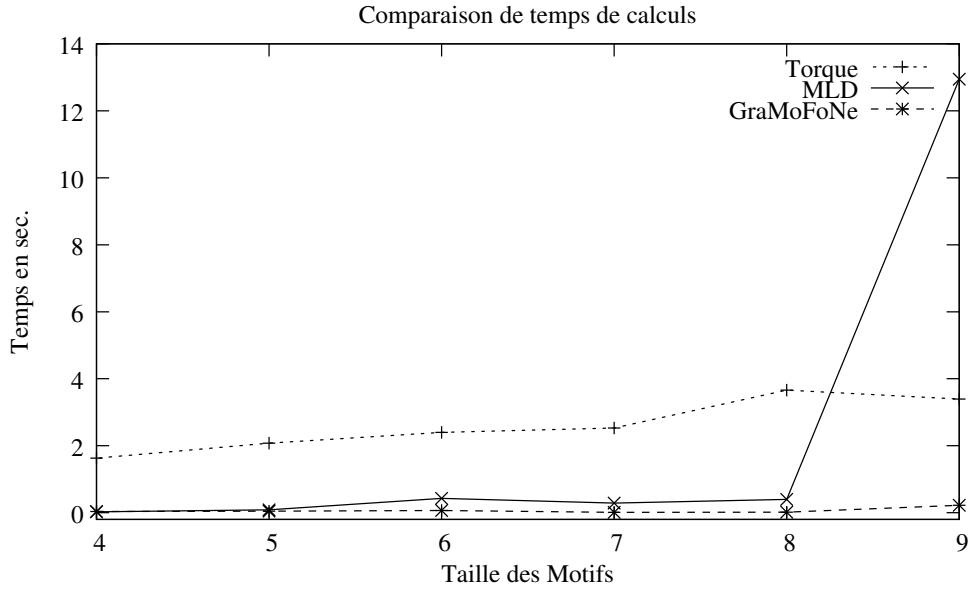


FIGURE 6.4 – Comparaison des temps de calculs moyens entre l'implémentation de la Proposition 4.5.2 (MLD), GraMoFoNe et Torque. Pour chaque taille donnée, la valeur calculée correspond au temps moyen pris par l'algorithme pour tous les motifs de cette taille pour l'ensemble des espèces dans tous les réseaux (sauf celui correspondant à l'espèce du motif).

On note que MLD doit effectuer exactement le même nombre d'opérations, qu'il existe une solution ou non. En effet, il faut évaluer le circuit arithmétique pour tous les sous-ensembles du motif selon la construction de la Proposition 4.5.2. Cependant, un solveur pseudo-booléen peut trouver rapidement qu'il n'existe pas certaines solutions, par exemple avec des contraintes rapidement non satisfaites.

Concernant GraMoFoNe, en autorisant des insertions et des délétions, déterminer s'il existe un résultat s'effectue en quelques secondes (5 à 20 pour des petits motifs, 40 à 60 pour des plus grands). Cependant, déterminer la meilleure solution peut prendre jusqu'à plusieurs minutes. L'utilisation d'un solveur pseudo-booléen externe empêche toutefois de prédire facilement les temps de calculs.

Ces résultats doivent être considérés avec précaution. En effet, MLD et GraMoFoNe sont écrits en Java, tandis que Torque est écrit en Python – le choix du langage peut en effet légèrement fausser les temps de calculs. De plus, l'implémentation de la Proposition 4.5.2 compte l'ensemble des solutions.

On peut cependant remarquer que l'approche par détection de monômes multilinéaires semble être utilisable en pratique, puisque les temps de calculs sont plus ou

moins comparables. On pourrait envisager l'implémentation des Propositions 5.1.2 et 5.1.3 afin de pouvoir gérer plus de cas biologiquement réalistes. Ceci implique l'implémentation délicate du Théorème 1.3.1 afin de résoudre le problème correspondant.

Chapitre 7

Comparaison de réseaux hétérogènes

Contenu

Ce court chapitre est basé sur un travail effectué en collaboration avec Guillaume Blin, Guillaume Fertin, Irena Rusu, Hafedh Mohamed-Babou et Stéphane Vialette, publié dans [BFMB+11]. Il s'agit toujours d'une étude concernant les réseaux biologiques, mais ici, nous ne cherchons pas de sous-graphes correspondant à un motif. Nous sommes cette fois-ci proches des problèmes d'intégrations, où étant donnés deux réseaux biologiques de types différents (ici un réseau d'interaction entre protéines et un réseau métabolique), on cherche des ensembles de sommets intéressants biologiquement dans les deux réseaux.

Nous décrivons brièvement les motivations et les définitions des problèmes associés. Ensuite, nous donnons un panorama des résultats de complexité obtenus avant de détailler certaines preuves.

7.1 Motivations et définitions

Nous avons déjà évoqué les motivations de la recherche de réseaux de types différents (hétérogènes) dans la Section 3.2.2. La plupart des méthodes existantes sont manuelles, ou effectuées au cas par cas [LDAM04, BML+05]. Dans [MBZS08, GSS10, GKSGBJ11], les auteurs effectuent une orientation des arêtes du réseau d'interactions entre protéines comme suit : étant donnée une liste de paires de sommets source-cible (s, t), le graphe est orienté de manière à maximiser le nombre de paires (s, t) telles qu'un

chemin orienté existe entre la source s et la cible t. L'avantage de cette méthode est qu'elle permet de revenir à un problème de comparaison entre deux réseaux homogènes. Cependant, trouver une telle orientation est un problème NP-Complet et difficile à approximer à ratio constant [MBZS08]. De plus, le bon comportement de cette méthode est dépendant du choix de la liste de paires source-cible. Par exemple, dans [GKSGBJ11], cette liste est construite manuellement et demande donc une expertise.

Récemment, un travail unifiant la comparaison et l'analyse de réseaux hétérogènes a été proposé dans [FMBR11]. Le problème central étudié dans [FMBR11] est le suivant : étant donnés un DAG D' et un graphe non-orienté G' utilisant le même ensemble de sommets, trouver le plus long chemin orienté dans D' dont les sommets induisent une composante connexe dans G'. Pour ce problème, D' et G' représentent respectivement un réseau métabolique et un réseau d'interactions entre protéines. Cependant, en toute généralité, les réseaux métaboliques ne sont pas des DAG et peuvent contenir des cycles.

Par conséquent, on s'intéresse dans la suite de cette section au problème suivant, appelé k-DAGCC (pour k DAG and Connected Component), pouvant être vu comme une manière d'effectuer un pré-traitement des graphes utilisés en entrée dans [FMBR11] : étant donnés un graphe orienté D et un graphe non-orienté G utilisant un même ensemble de sommets V et un entier k, existe-t-il un ensemble de sommets $V_1, V_2, \ldots, V_{k'}$, $k' \leq k$, tel que, pour chaque $1 \leq i \leq k'$, (i) $D[V_i]$ soit un DAG et (ii) $G[V_i]$ soit connexe ? La méthode employée pour pouvoir utiliser un même ensemble de sommets est décrite dans [FMBR11]. Nous étudions dans la suite deux variantes du problème k-DAGCC : (i) la version où les $V_i, 1 \leq i \leq k'$ doivent former une *partition* de V et (ii) la version où les $V_i, 1 \leq i \leq k'$ doivent former une *couverture* de V.

Une autre motivation pour les deux variantes du problème k-DAGCC se trouve dans la recherche de voies métaboliques correspondant fonctionnellement à des protéines du réseau d'interactions entre protéines. En effet, une voie métabolique correspond à un DAG [PRYLZU05], tandis qu'un ensemble de protéines codant une fonction biologique correspond à un sous-graphe connexe [BML$^+$05, SSK$^+$05]. Par conséquent, il est intéressant d'extraire de l'information biologiquement pertinente depuis deux réseaux larges en des sous-réseaux plus petits qui (i) portent une information biologique et (ii) sont plus simples à interpréter (pour le problème étudié dans [FMBR11]).

On propose dans la suite un panorama des résultats de complexité publiés dans [BFMB$^+$11] pour les deux variantes du problème, selon les contraintes imposées aux entrées. Plus précisément, nous avons obtenu plusieurs algorithmes de complexité polynomiale. Malheureusement, de tels algorithmes ne sont possibles que sous de fortes contraintes algorithmiques ; le problème est en effet NP-Complet et difficile à approximer, même avec d'importantes restrictions.

Nous rappelons que les deux graphes (orientés et non-orientés) partagent le même ensemble de sommets. Par conséquent, on note $D = (V, A)$ le graphe orienté et $G = (V, E)$ le non-orienté. Les deux problèmes étudiés, k-DAGCC-PARTITION et k-DAGCC-

COVER sont définis comme suit.

k-DAGCC-PARTITION

• **Entrée** : Un graphe orienté $D = (V, A)$, un graphe non-orienté $G = (V, E)$, un entier k.

• **Sortie** : Une partition $\mathcal{P} = \{V_1, V_2, \ldots, V_{k'}\}$ de V telle que (i) $k' \leq k$, (ii) $\forall 1 \leq i \leq k', D[V_i]$ soit un DAG et (iii) $\forall 1 \leq i \leq k', G[V_i]$ soit connexe.

k-DAGCC-COVER

• **Entrée** : Un graphe orienté $D = (V, A)$, un graphe non-orienté $G = (V, E)$, un entier k.

• **Sortie** : Une couverture $\mathcal{C} = \{V_1, V_2, \ldots, V_{k'}\}$ de V telle que (i) $k' \leq k$, (ii) $\forall 1 \leq i \leq k', D[V_i]$ soit un DAG et (iii) $\forall 1 \leq i \leq k', G[V_i]$ soit connexe.

Les versions naturelles de minimisation de ces problèmes sont notées respectivement MINIMUM DAGCC-PARTITION et MINIMUM DAGCC-COVER.

7.2 Résultats de complexité

Un ensemble de résultats est obtenu pour les deux versions du problème, que l'on synthétise dans les Tables 7.1 et 7.2. On donne dans la suite de la section les détails de quelques preuves seulement – l'ensemble des preuves des autres résultats se trouve dans [BFMB+11].

k \ G	Graphe	Outerplanar	Arbre	Étoile	Chemin
$(n-k) = \mathcal{O}(1)$	P				
$k = \mathcal{O}(1)$	NP-Complet	P			
k non borné	Inapprox. à moins de $n^{1-\epsilon}$	APX-Difficile		NP-Complet	P

TABLE 7.1 – Synthèse des résultats de complexité obtenus dans [BFMB+11] pour les problèmes k-DAGCC-PARTITION et MINIMUM DAGCC-PARTITION.

On remarque à la vue de ces tables que fixer le nombre de parties (c'est-à-dire quand k est une constante) permet d'obtenir des algorithmes polynomiaux quand G est d'une topologie plus simple qu'un graphe. Cependant, le problème devient difficile, même pour des topologies simples telles que l'étoile quand k n'est pas borné.

k \ G	Graphe	Outerplanar	Arbre	Étoile	Chemin
$k = \mathcal{O}(1)$	NP-Complet				P
k non borné	Inapproximable à moins de $n^{1-\epsilon}$				P

TABLE 7.2 – Synthèse des résultats de complexité obtenus dans [BFMB+11] pour les problèmes k-DAGCC-COVER et MINIMUM DAGCC-COVER.

Néanmoins, quand la solution doit être une couverture de V et non plus une partition, le problème devient plus difficile, même lorsque k est une constante.

Existe t-il un algorithme de complexité paramétrée pour k-DAGCC-PARTITION lorsque G est un arbre ? Le problème est en effet W[1]-Difficile lorsque G est un graphe car il est NP-Complet pour un k fixé.

Donnons maintenant les détails de trois preuves en particulier. On rappelle qu'étant donnés un graphe orienté $D = (V, A)$ et un graphe non orienté $G = (V, E)$, il est possible de tester si D est un DAG (resp. si G est connexe) en temps $\mathcal{O}(|V| + |A|)$ (resp. $\mathcal{O}(|V| + |E|)$) avec un algorithme de parcours en profondeur [CLRS02].

Proposition 7.2.1. *Il existe un algorithme polynomial ayant une complexité en temps de* $\mathcal{O}(|V|(|V| + |A|))$ *pour le problème* MINIMUM DAGCC-PARTITION *si* G *est un chemin.*

Démonstration. La preuve utilise un simple algorithme glouton. Le chemin G à n sommets est écrit $G = (v_1, v_2, \ldots, v_n)$. On considère les sommets de G ordonnés de v_1 à v_n.

On débute l'algorithme avec un ensemble vide $S = \emptyset$. Pour chaque sommet v_i, si $D[S \cup \{v_i\}]$ est un DAG, on ajoute v_i à S, sinon on indique que S est un élément de la partition recherchée et on ré-initialise S à $S = \{v_i\}$. Lorsque l'algorithme se termine (c'est-à-dire lorsque tous les sommets de G ont été considérés), on indique que l'ensemble S courant est le dernier élément de la partition recherchée. Cet algorithme glouton est clairement polynomial, puisque pour les $|V|$ sommets, on effectue une recherche de DAG en temps $\mathcal{O}(|V| + |A|)$.

On voit facilement que cet algorithme glouton produit une partition de V (chaque sommet ne peut faire partie que d'un seul S). Il reste à prouver que cette partition est de taille minimum. Soit $\mathcal{P} = \{V_1, V_2, \ldots V_k\}$ la partition retournée par l'algorithme décrit. Par contradiction, supposons qu'il existe une partition $\mathcal{P}' = \{V'_1, V'_2, \ldots, V'_\ell\}$ de taille strictement inférieure, donc où $\ell < k$. Premièrement, on observe que par construction, V_1 ne peut pas être strictement inclus dans V'_1. Ensuite, puisque chaque sous-graphe

connexe de G est constitué de sommets ayant des indices consécutifs, il existe deux entiers $2 \leq i < k$ et $2 \leq j \leq \ell$ tels que (i) V_i est strictement inclus dans V'_j et (ii) V'_j contient le premier sommet (noté x) de V_{i+1}. Ceci mène à une contradiction puisque, par construction, $D[V_i \cup \{x\}]$ n'est pas un DAG (voir aussi Figure 7.1).

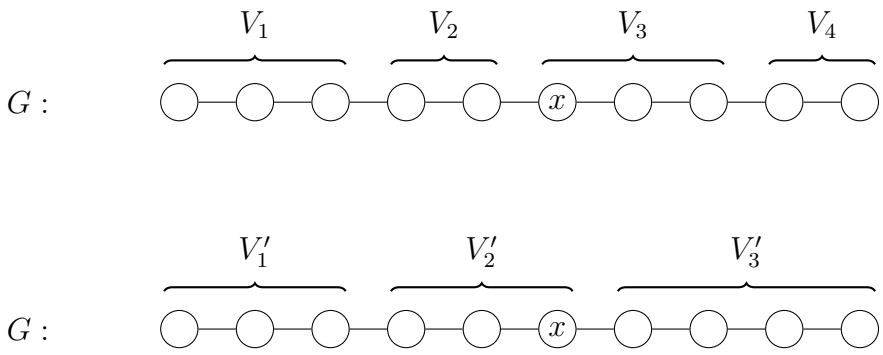


FIGURE 7.1 – Étant donné le graphe G qui est un chemin, on représente la partition $\mathcal{P} = \{V_1, V_2, V_3, V_4\}$ comme étant celle retournée par l'algorithme glouton. Supposons qu'il existe une partition de taille strictement inférieure $\mathcal{P}' = \{V'_1, V'_2, V'_3\}$. Alors, il existe deux entiers $2 \leq i < k$ et $2 \leq j \leq \ell$ (ici $i = j = 2$) tels que (i) V_i est strictement inclus dans V'_j et (ii) V'_j contient le premier sommet (noté x) de V_{i+1}. Par construction de $\mathcal{P}$, $V_2 \cup \{x\}$ contient un cycle dans D, donc V'_2 n'induit pas un DAG et $\mathcal{P}'$ n'est pas une partition valide.

□

Proposition 7.2.2. *Le problème* k-DAGCC-PARTITION *est* NP*-Complet, même quand* G *est une étoile.*

Démonstration. On voit facilement que le problème est dans NP car étant donné un ensemble $\mathcal{P} = \{V_1, V_2, \ldots, V_k\}$, on peut vérifier en temps polynomial si $\mathcal{P}$ est une partition, si $D[V_i]$ est un DAG et si $G[V_i]$ est connexe, pour chaque $1 \leq i \leq k$. Pour prouver que le problème est en plus NP-Difficile, on propose une réduction depuis le problème de décision INDEPENDENT SET où la solution est au minimum de taille k' (nommé ici k'-IS), connu NP-Complet [GJ79].

Depuis une instance $\mathcal{I} = (G_I = (V_I, E_I), k')$ du problème k'-IS, on construit une instance $\mathcal{I}' = (D = (V, A), G = (V, E), k)$ du problème k-DAGCC-PARTITION comme suit :

- $V = V_I \cup \{v_r\}$,
- $A = \{(v, v'), (v', v) : \{v, v'\} \in E_I\}$,
- $E = \{\{v_r, v\} : v \in V_I\}$,
- $k = |V_I| - k' + 1$.

De manière informelle, le graphe orienté D est obtenu depuis G_I en remplaçant chaque arête par deux arcs de directions opposées et en ajoutant un sommet isolé v_r. Le graphe non-orienté G est lui une étoile, donc le centre est v_r. Par conséquent, le sommet v_r appartient à tout sous-graphe connexe de G ayant une taille strictement supérieure à 1. La construction de ces deux graphes est illustrée Figure 7.2.

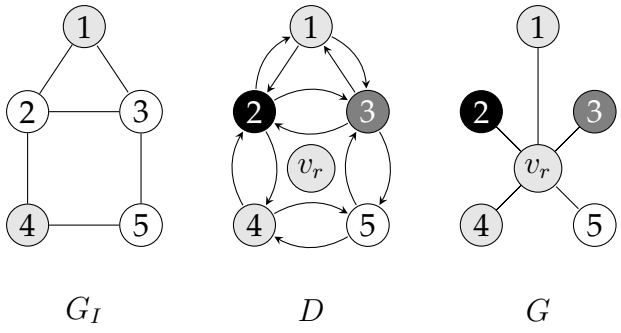


FIGURE 7.2 – Illustration de la construction des graphes D et G étant donné le graphe G_I. Une solution possible pour le problème 2-IS est mise en évidence (sommets $\{1, 4\}$) ainsi que la 4-partition de V correspondante.

On montre maintenant qu'étant donnée une solution de taille k' pour une instance $\mathcal{I}$ du problème k'-IS, il existe une solution de taille $k = |V_I| - k' + 1$ pour l'instance $\mathcal{I}'$ du problème k-DAGCC-PARTITION. Premièrement, étant donnée une solution pour $\mathcal{I}$ de taille supérieure ou égale à k', on peut choisir n'importe quel sous-ensemble V'_I de k' sommets de cette solution (un sous-ensemble d'un ensemble indépendant reste un ensemble indépendant). Ensuite, on construit comme suit la partition $\mathcal{P} = \{V_1, V_2 \dots V_k\}$ de V, où $k = |V_I| - k' + 1$. On construit $V_1 = V'_I \cup \{v_r\}$. Ensuite, étant donné n'importe quel ordre fixé sur les sommets de $V_I \setminus V'_I$ et pour chaque $2 \leq i \leq k$, on construit $V_i = \{v\}$, où v est le $(i-1)^{eme}$ sommet de cet ordre fixé. On note tout d'abord que chaque $D[V_i]$, $2 \leq i \leq k$ est bien un DAG car il est constitué d'un seul sommet. De plus, par définition, $\nexists \{v, v'\} \subseteq V'_I$ tels que $\{v, v'\} \in E_I$. Par conséquent, $D[V_1]$ est constitué de sommets isolés (et est donc également un DAG). Les sous-graphes $G[V_i]$, $2 \leq i \leq k$, sont connexes, tandis que la connexité de $G[V_1]$ est assurée par le fait que $v_r \in V_1$.

Inversement, on montre maintenant qu'étant donnée une solution de taille k pour l'instance $\mathcal{I}'$ du problème k-DAGCC-PARTITION, il existe une solution de taille $k' = |V_I| - k + 1$ pour l'instance $\mathcal{I}$ du problème k-IS. Étant donnée une solution $\mathcal{P} = \{V_1, V_2 \dots V_k\}$ pour $\mathcal{I}'$, on construit une solution de taille $|V'_I| = |V_I| - k + 1$ avec $V'_I = V_j \setminus \{v_r\}$, où $D_j = (V_j, A_j)$ est le DAG contenant v_r. Comme mentionné précédemment, chaque sous-graphe connexe de G dans la solution de taille strictement supérieure à 1 doit contenir le sommet v_r. Selon les contraintes du problème k-IS, pour chaque $1 \leq i < j \leq k$, on a $V_i \cap V_j = \emptyset$. Donc, au plus un des ensembles V_i peut être de taille strictement supérieure à 1 et celui-ci contient v_r. Par conséquent, $|V'_I| = |V_j| - 1 = ((|V_I| + 1) - (k-1)) - 1$, c'est-à-dire $|V'_I| = |V_I| - k + 1$. De plus, V'_I est effectivement une solution pour $\mathcal{I}$ car comme D_j est un DAG, il ne contient pas de cycle. □

Proposition 7.2.3. *Le problème* MINIMUM DAGCC-PARTITION *est* APX*-Difficile, même quand G est un arbre.*

Démonstration. On propose une L-réduction depuis le problème APX-Difficile MINIMUM SET COVER-2 [PY91], un cas particulier du problème MINIMUM SET COVER dont

on rappelle la définition.

MINIMUM SET COVER-2

• **Entrée** : Un ensemble $X = \{x_1, x_2, \ldots, x_{|X|}\}$, une collection $\mathcal{S} = \{S_1, S_2, \ldots, S_{|\mathcal{S}|}\}$ de sous-ensembles de X où chaque élément de X apparaît au plus deux fois.

• **Sortie** : Un sous-ensemble $\mathcal{S}' \subseteq \mathcal{S}$ tel que chaque élément de X apparaisse dans un moins un membre de $\mathcal{S}'$.

• **Mesure** : La taille de $\mathcal{S}'$.

Depuis n'importe quelle instance $\mathcal{I} = (X, \mathcal{S})$ du problème MINIMUM SET COVER-2, on construit une instance $\mathcal{I}' = (D = (V, A), G = (V, E))$ pour le problème MINIMUM DAGCC-PARTITION de la manière suivante.

- $V = \{v_r\} \cup \{v_i : S_i \in \mathcal{S}\} \cup \{v_i^k : S_i \in \mathcal{S}, x_k \in S_i\}$,
- $A = \{(v_i^k, v_j^k), (v_j^k, v_i^k) : x_k \in S_i \cap S_j\} \cup \{(v_r, v_i^k), (v_i^k, v_r) : \nexists S_j \text{ t.q. } x_k \in S_i \cap S_j\}$,
- $E = \{\{v_r, v_i\} : S_i \in \mathcal{S}\} \cup \{(v_i, v_i^k) : x_k \in S_i, S_i \in \mathcal{S}\}$.

En d'autres termes, G est un arbre enraciné en v_r, dont les fils sont les sommets $v_i, 1 \leq i \leq |\mathcal{S}|$ (où v_i, $1 \leq i \leq |\mathcal{S}|$, représente S_i). Chaque sommet v_i a un fils par élément de S_i (v_i^k pour $x_k \in S_i$). Concernant D, les sommets $v_i, 1 \leq i \leq |\mathcal{S}|$, sont isolés, tandis que deux sommets (v_i^k, v_j^k), représentant les deux occurrences d'un élément x_k dans $\mathcal{S}$, forment un cycle de longueur deux. Enfin, chaque sommet restant forme un cycle de longueur deux avec le sommet v_r. Une illustration de la construction de l'instance $\mathcal{I}'$ depuis $\mathcal{I}$ se trouve Figure 7.3.

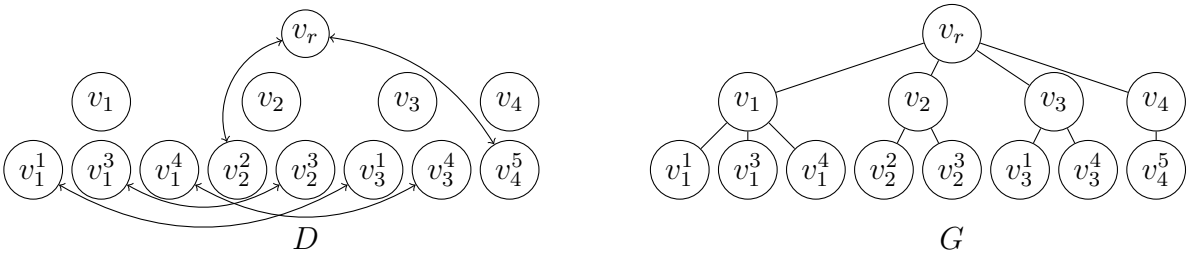


FIGURE 7.3 – Illustration de la construction des graphes D et G, depuis $\mathcal{S} = \{\{x_1, x_3, x_4\}, \{x_2, x_3\}, \{x_1, x_4\}, \{x_5\}\}$.

On montre dans un premier temps que depuis une solution de taille k pour une instance $\mathcal{I}$ de MINIMUM SET COVER-2, on peut construire une solution de taille inférieure à $k + 1$ pour l'instance $\mathcal{I}'$ du problème MINIMUM DAGCC-PARTITION. On considère donc une solution $\mathcal{S}' \subseteq \mathcal{S}$ de taille k nous permettant de construire une partition $\mathcal{P} = \{V_1, V_2 \ldots, V_k, V_{k+1}\}$ de la manière suivante. Pour chaque $S_i \in \mathcal{S}'$, $1 \leq i \leq k$, on construit $V_i = \{v_i\} \cup \{v_i^k : x_k \in S_i\}$ et on construit $V_{k+1} = \{v_r\} \cup \{v_j, v_j^k : x_k \in S_j, S_j \in \mathcal{S} \setminus \mathcal{S}'\}$. En d'autre termes, il existe un ensemble V_i (donnant un sous-arbre de G enraciné en

v_i) pour chaque S_i de la solution $\mathcal{S}'$, tandis que V_{k+1} contient les sommets restants (y compris la racine v_r de G). Par conséquent et par construction, pour chaque $1 \leq i \leq k+1$, $G[V_i]$ est bien connexe. De plus, chaque $D[V_i]$, $1 \leq i \leq k$, est un DAG car il ne contient pas v_r et que chaque élement de X apparaît au plus une fois dans S_i. Enfin, $D[V_{k+1}]$ est également un DAG car, par définition de MINIMUM SET COVER-2, chaque élément de X apparaît au plus deux fois dans $\mathcal{S}$ et au moins une fois dans $\mathcal{S}'$. Par conséquent, dans $\mathcal{S} \setminus \mathcal{S}'$, chaque élément apparaît au plus une fois et il n'existe aucun cycle dans $D[V_{k+1}]$.

Inversement, on montre maintenant qu'étant donnée une solution de taille k pour l'instance $\mathcal{I}'$ du problème MINIMUM DAGCC-PARTITION, on trouve une solution de taille $k-1$ pour l'instance $\mathcal{I}$ du problème MINIMUM SET COVER-2. On considère une solution valide pour $\mathcal{I}'$ que l'on note $\mathcal{P} = \{V_1, V_2 \dots V_k\}$. On présume sans perte de généralité que $v_r \in V_1$. On construit alors une solution pour $\mathcal{I}$ telle que $\mathcal{S}' = \{S_i : \exists v_i^j \notin V_1\}$. Autrement dit, on ajoute dans la solution tous les ensembles ayant un élément correspondant à un sommet qui n'appartient pas à V_1.

Pour prouver que la taille de $\mathcal{S}'$ est correcte, on observe que par construction, il existe au plus un sommet de $\{v_i : 1 \leq i \leq |\mathcal{S}|\}$ dans chaque $G[V_j]$, $2 \leq j \leq k$, car chaque chemin reliant deux sommets de ce type passe par v_r, qui est déjà considéré contenu dans V_1. Par conséquent, $|\mathcal{C}'| \leq k-1$. Il reste maintenant à prouver que $\mathcal{S}'$ est bien une solution pour $\mathcal{I}$. Pour cela, considérons chaque élément $x_i \in X$. Si x_i apparaît exactement une fois dans $\mathcal{S}$ (sans perte de généralité, on suppose que $x_i \in S_j$), alors le sommet v_j^i correspondant ne peut pas appartenir à V_1 car par construction, il induit un cycle avec v_r dans $D[V_1]$. Par conséquent, $S_j \in \mathcal{S}'$. Sinon, x_i apparaît exactement deux fois, et alors au plus un des deux sommets v_j^i et $v_{j'}^i$ correspondants appartient à V_1, car sinon cela induirait un cycle dans $D[V_1]$. Donc, chaque $x_i \in X$ apparaît au moins une fois dans $\mathcal{C}'$.

La réduction présentée est bien une L-réduction depuis le problème MINIMUM SET COVER-2 vers le problème MINIMUM DAGCC-PARTITION. Comme le problème MINIMUM SET COVER-2 est APX-Difficile [PY91], le problème MINIMUM DAGCC-PARTITION l'est également. □

7.3 Ajouter une contrainte de connexité sur les DAG

Dans les constructions des preuves de difficulté présentées dans [BFMB+11] pour le problème k-DAGCC, on observe que la plupart des sous-graphes $D[V_i]$, $1 \leq i \leq k'$ construits ne sont pas (faiblement) connexes (contrairement à $G[V_i]$ qui est connexe par définition). Ceci peut-être interprété comme n'étant pas biologiquement pertinent.

C'est pourquoi nous avons introduit le problème k-DAGCCC (pour k DAG Connected and Connected Component). Étant donnés un graphe orienté D et un graphe non-orienté G utilisant un même ensemble de sommets V et un entier k, existe t-il un ensemble de sommets $V_1 \dots V_{k'}$, $k' \leq k$, tel que, pour chaque $1 \leq i \leq k'$, (i) $D[V_i]$ est

un DAG, (ii) $D[V_i]$ est faiblement connexe et (iii) $G[V_i]$ est connexe ? Nous pouvons également considérer les deux variantes du problème (la version où les $V_i, 1 \leq i \leq k'$ doivent former une partition de V et celle où elles doivent former une couverture de V). Si la plupart des résultats restent valides pour la recherche d'une partition, ce n'est pas le cas pour la recherche d'une couverture.

Troisième partie

Génomique comparative et structure secondaire d'ARN

Chapitre 8

Quelques problèmes concernant la génomique comparative

Contenu

On trouve dans ce chapitre quatre problèmes ayant trait à la génomique comparative. Dans la Section 8.1, nous considérons uniquement les différences entre les gènes, puis, nous cherchons à retrouver les ancêtres de la population considérée. Ensuite, nous comparons des génomes entiers. Plus précisément, on cherche dans la Section 8.2 à choisir un exemplaire de chaque gène d'un génome afin d'optimiser une certaine distance, puis, dans la Section 8.3, on cherche à retrouver un ensemble de gènes ancestraux parmi plusieurs génomes actuels. Enfin, dans la Section 8.4, le but est de réconcilier un ensemble d'arbres phylogénétiques spécifiques en minimisant les duplications de gènes.

Pour chacun de ces problèmes, nous exposons les motivations avant de rappeler les résultats connus. Ensuite, nous présentons nos contributions sans en donner les preuves. Ces résultats sont, dans les quatre cas, des résultats de difficulté (NP-Complétude ou difficulté d'approximation) et sont à chaque fois contre-balancés par des résultats positifs (algorithme polynomial ou de complexité paramétrée).

8.1 La mosaïque minimum

On présente dans cette section le travail effectué en collaboration avec Guillaume Blin, Romeo Rizzi et Stéphane Vialette, publié en 2011 dans [BRSV11a] (la version longue est acceptée pour publication [BRSV11b]). La suite de cette section est consacrée au problème consistant à retrouver des événements de recombinaisons [KG98, Ukk02, KRU04, RU07, WG07]. Après avoir présenté les motivations et les travaux connus, nous présentons un résultat de difficulté, contre-balancé par plusieurs résultats positifs, en l'occurrence des algorithmes polynomiaux lorsque des informations supplémentaires sont disponibles.

8.1.1 Motivations et définitions

Étant données deux personnes n'étant pas de la même famille, leur séquence ADN ne contient qu'environ 0.1% de différences. Ces petites variations apparaissent à des positions spécifiques, nommées *Single Nucleotide Polymorphisms* (SNPs). Sur ces positions, les deux nucléotides (de deux membres d'une même population ou de la paire de chromosomes d'un même individu) sont différents (voir Figure 8.1).

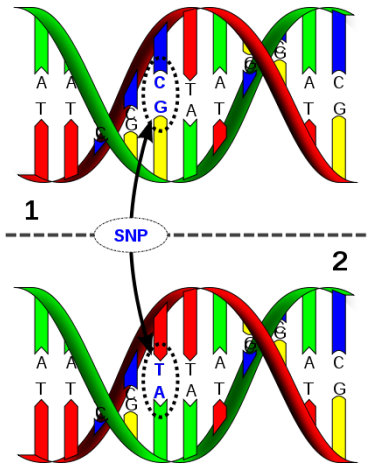


FIGURE 8.1 – Illustration d'un SNP. Source Wikipedia.

Cette faible différence génétique est particulièrement importante car elle est responsable de la majorité des différences génétiques et explique en partie pourquoi certaines personnes réagissent différemment aux médicaments ou ont des risques différents de maladie. De plus, ces données sont plus économiques à collecter que le génome complet car de taille moindre. C'est pourquoi il est important de découvrir les variantes de l'ADN contribuant à des risques de maladies communes.

Dans la plupart des SNPs, seulement deux nucléotides sur les quatre possibles apparaissent. Par conséquent, les séquences de SNPs sont représentées par des séquences binaires (0 ou 1) de même longueur, où chaque caractère représente une des deux possibilités fréquentes.

Les variations génétiques entre les espèces sont principalement induites par un processus nommé *recombinaison*. Étant données deux séquences de taille identique, une recombinaison génère une troisième séquence de même taille, en concaténant un préfixe de la première séquence avec un suffixe de la seconde séquence [KRU04]. L'indice de la séquence construite où la jointure est faite entre le préfixe et le suffixe est appelé *breakpoint*. Une illustration de la recombinaison est donnée Figure 8.2.

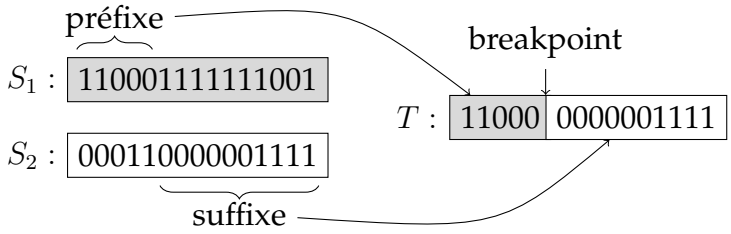


FIGURE 8.2 – Recombinaison de deux séquences de SNPs binaires S_1 et S_2, permettant l'obtention de T.

Retrouver les événements de recombinaisons est devenu un problème central en bio-informatique [KG98, Ukk02, KRU04, RU07, WG07]. Dans la plupart des modèles considérés, une population actuelle est supposée descendante d'un faible nombre de séquences spécifiques, appelées *founders*. Une séquence actuelle est considérée comme construite de blocs, provenant des *founders*, suite à une série de recombinaisons (voir Figure 8.3). Le terme de *mosaïque* est souvent employé pour décrire la structure d'ADN induite par les recombinaisons. En effet, on obtient une sorte de mosaïque si, après avoir assigné une couleur pour chaque *founder*, on donne à chaque bloc de la population courante la couleur du *founder* dont le bloc provient (voir Figure 8.4).

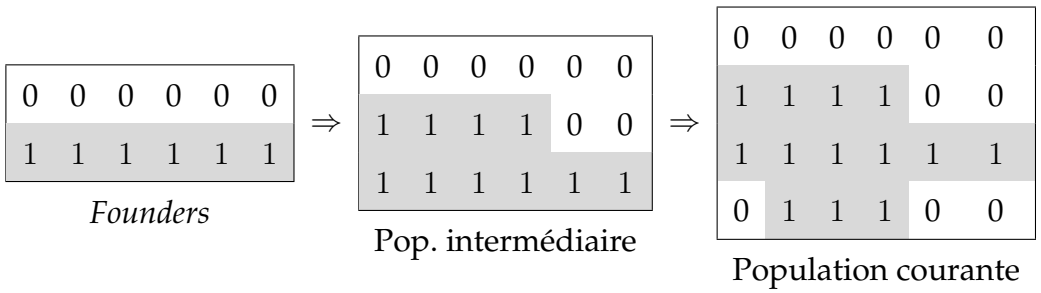


FIGURE 8.3 – Une suite de trois recombinaisons depuis les *founders* jusqu'à la population actuelle, formée de blocs. La population courante contient alors trois breakpoints (passages de blocs sombres aux blocs clairs).

Une question liée à la recherche de *founders* est le problème MINIMUM MOSAIC. Il consiste à trouver les *founders* permettant de reconstruire la population actuelle, en produisant le minimum de breakpoints. Plus formellement, le problème (donné sous sa forme de décision) est défini de la manière suivante :

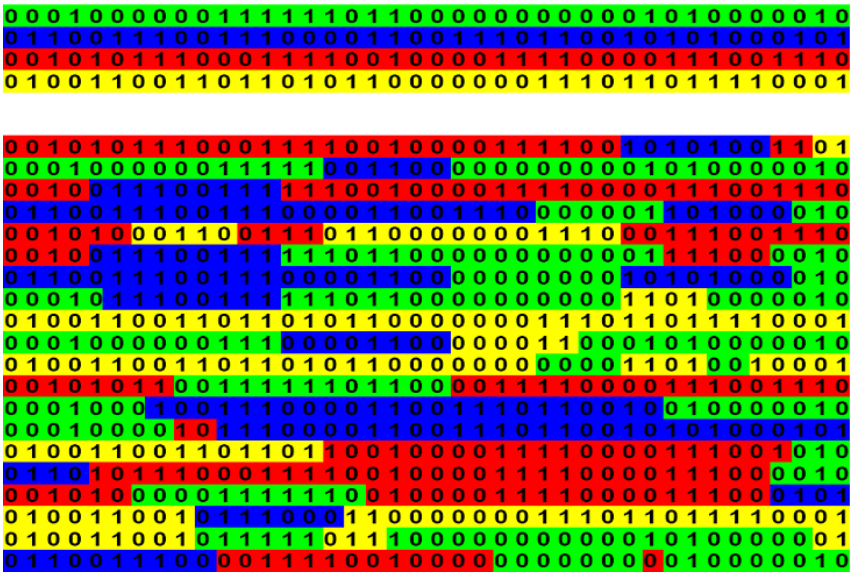


FIGURE 8.4 – Illustration générée par RecBlock [WG07]. Chaque *founder* porte une couleur différente. La population courante est constituée de blocs portant la couleur du *founder* dont il provient.

MINIMUM MOSAIC

• **Entrée** : Un ensemble de m séquences de longueur n (la population actuelle), un entier K.

• **Sortie** : Un ensemble de K *founders* induisant le minimum de breakpoints.

Sur l'exemple Figure 8.3, les 4 séquences de longueur 6 de la population courante sont à droite de la figure. Choisir comme *founders* les 2 séquences à gauche de la figure induit trois breakpoints (il y a trois alternances entre blocs blancs et noirs).

8.1.2 Résultats connus

Le problème est formulé pour la première fois par Ukkonen [Ukk02]. Ce dernier considère deux critères d'optimisations : le nombre de *founders* et le nombre de breakpoints. Il donne un algorithme ayant une complexité de $\mathcal{O}(mn)$ pour le problème MINIMUM MOSAIC où $K = 2$ (Wu et Gusfield donnent un résultat similaire dans [WG07]).

Pour résoudre le problème quand $K \geq 2$, Ukknonen [Ukk02] donne également un algorithme exact ayant une complexité de $\mathcal{O}(nK^{2m})$ par programmation dynamique (la complexité n'est pas donnée explicitement, mais nous l'avons justifiée dans [BRSV11a]).

Dans [RU07], Rastas et Ukkonen considèrent le problème MINIMUM MOSAIC où sont autorisées des mutations, c'est-à-dire des différences caractère par caractère entre les *founders* et les séquences actuelles. Plus précisément, pour chaque séquence actuelle, un score de $k + k'c$ est calculé, où k est le nombre de breakpoints, k' le nombre de

mutations et c un coût pénalisant une mutation. Ils prouvent que ce problème est NP-Complet, même pour $K = 2$. Cependant, on note que ce résultat est un peu artificiel car, dans la preuve, la pénalité fixée à $c = \frac{1}{nm}$ implique une interdiction des recombinaisons. En effet, effectuer nm mutations est une opération aussi coûteuse qu'un seul événement de recombinaison impliquant un breakpoint. C'est d'ailleurs la raison pour laquelle il existe un algorithme polynomial pour ce problème sans mutation quand $K = 2$ [Ukk02, WG07].

Enfin, des heuristiques ainsi que des bornes sur le nombre minimum de breakpoints ont été proposées pour le problème MINIMUM MOSAIC [RB09, Wu10].

Finalement, la complexité du problème MINIMUM MOSAIC est inconnue lorsque le nombre de *founders* K est strictement supérieur à 2.

8.1.3 Contributions

Nous avons prouvé dans [BRSV11a] que le problème MINIMUM MOSAIC est NP-Complet si K est non borné et sans utiliser un coût de mutation non réaliste comme dans [RU07].

Pour rendre plus aisée la réduction, on prouve dans un premier temps que si le problème MINIMUM MOSAIC est NP-Difficile avec des séquences utilisant un alphabet arbitraire, alors le problème l'est aussi sur un alphabet binaire. Ceci nous permet d'utiliser un alphabet de plus grande taille pour la réduction. L'idée est simplement d'encoder les $|\Sigma|$ caractères de l'alphabet arbitraire par des mots de taille $\lceil \log_2 k \rceil$. Par exemple, si l'alphabet est $\Sigma = \{A, B, C\}$, on peut transformer les A par le mot 00, les B par 01 et les C par 10. On prouve ensuite que depuis une solution utilisant Σ, il existe une solution utilisant un alphabet binaire avec le même nombre de breakpoints et inversement [BRSV11a].

De plus, on ajoute la possibilité qu'un nombre fixé $K' < K$ de *founders* fasse partie de la solution (que l'on nomme *founders forcés*). On peut en effet réduire en temps polynomial une instance contenant des *founders forcés* à une instance n'en contenant pas. Pour chaque *founder forcé* f^f, on ajoute $n\,m$ copies de f^f parmi les séquences d'entrée. Par la suite, si la séquence f^f n'est pas dans la solution, alors elle provoque au moins un breakpoint. Vu son nombre de répétitions, elle provoque alors nm breakpoints, ce qui correspond au nombre maximum de breakpoints que l'on peut obtenir.

Grâce à l'utilisation d'un alphabet arbitraire et d'un certain nombre de *founders forcés*, nous sommes capable d'effectuer une réduction depuis le problème VERTEX COVER.

Proposition 8.1.1. *Le problème* MINIMUM MOSAIC *est* NP*-Complet lorsque K est non borné.*

Afin de contre-balancer la difficulté de ce problème, nous avons également obtenu un ensemble d'algorithmes exacts, pouvant être considérés comme polynomiaux si certaines informations supplémentaires sont données dans l'instance du problème. Leur complexité respective est donnée dans la Table 8.1

Info. connue	Rien	Emplacement des B	Nombre de B pour chaque séquence	Nombre de B
Complexité	$\mathcal{O}(nK^{2m})$	$\mathcal{O}(\|B\|Kn)$	$\mathcal{O}(n^{\|B\|}\|B\|Kn)$	$\mathcal{O}((K+\|B\|)^{\|B\|}nK^{2\|B\|})$

TABLE 8.1 – Complexité de quatre algorithmes pour résoudre le problème MINIMUM MOSAIC, lorsque différentes informations supplémentaires sont disponibles (B correspond à breakpoint).

L'algorithme exact lorsqu'aucune information n'est considérée connue est présenté dans [Ukk02], mais la détermination de sa complexité est effectuée dans [BRSV11a].

L'idée pour obtenir un algorithme polynomial lorsque l'on connait l'emplacement de chaque breakpoint est de déterminer, par un algorithme glouton, l'emplacement dans la solution de chaque sous-séquence sans breakpoint de l'entrée (qui doit par définition apparaitre telle quelle dans la solution).

L'idée de l'algorithme polynomial si le nombre de breakpoints pour chaque séquence est connu (et donc que le nombre total de breakpoints est borné) consiste à envisager toutes les positions possibles de breakpoints et d'utiliser l'algorithme précédent.

Enfin, lorsque seul le nombre de breakpoints est connu, une étude du nombre de possibilités et l'utilisation de l'algorithme décrit dans [Ukk02] permet d'obtenir une telle complexité. Nous avons donné le détail de ces trois derniers algorithmes dans [BRSV11a].

Lorsque $K = 2$, le problème MINIMUM MOSAIC est polynomial. Nous avons montré que lorsque K n'était pas borné, le problème était NP-Complet. Une importante question ouverte concerne donc la complexité du problème lorsque K est borné.
De plus, la réduction proposée ne préserve pas le ratio d'approximation. Le problème MINIMUM MOSAIC admet-il un schéma d'approximation polynomial (PTAS) ?

8.2 Comparaison de génomes avec duplications

Cette section est consacrée à un travail effectué en collaboration avec Guillaume Blin, Guillaume Fertin et Stéphane Vialette, publié dans [BFSV09].

8.2.1 Motivations et définitions

Un problème important en bio-informatique consiste à comparer deux espèces et plus spécifiquement, à trouver des ensembles conservés de gènes dans leur génome res-

pectif. En effet, un ensemble de gènes conservés dans le même ordre durant l'évolution suggère une participation à un même processus biologique. Trouver de tels ensembles est habituellement effectué en optimisant une certaine mesure de similarité. Plusieurs mesures ont été étudiées : le nombre de breakpoints (si le terme est identique à celui utilisé dans la section précédente, la définition est différente), le nombre d'adjacences, d'intervalles conservés, d'intervalles communs... Dans la suite de cette section, on s'intéresse au calcul du *nombre de breakpoints* entre deux génomes.

On considère un génome comme une séquence de gènes signés (le signe du gène représente le brin d'ADN où se situe le gène). On appelle famille de gènes l'ensemble des gènes considérés comme issus d'un gène ancestral. Un génome est sans duplication s'il existe une seule occurrence de chaque famille de gènes. Le i^{eme} gène de G est noté $G[i]$. Pour un gène g, on note $\overline{g}$ le gène portant le signe opposé. Étant donné un génome G sans duplication, on dit qu'un gène $g = G[i]$ précède immédiatement $g' = G[j]$ si et seulement si $j = i + 1$. Intuitivement, un breakpoint est une adjacence non conservée. Plus précisément, étant donnés deux génomes G_1 et G_2 sans duplication, si un gène g précède immédiatement g' dans G_2 et qu'aucune des deux conditions suivantes n'est remplie :

- g précède immédiatement g' dans G_1,
- $\overline{g'}$ précède immédiatement $\overline{g}$ dans G_1,

alors g et g' forment un breakpoint dans G_1 (voir Figure 8.5). Le nombre de breakpoints entre G_1 et G_2 est alors défini comme le nombre de breakpoints dans G_2.

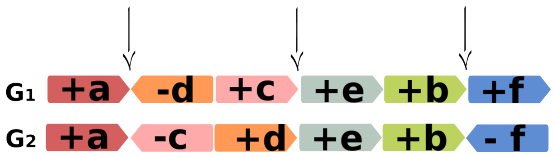


FIGURE 8.5 – Illustration de trois breakpoints (représentés par les flèches) pour les deux génomes G_1 et G_2.

Les mesures citées précédemment sont bien définies quand le génome ne contient pas de gènes dupliqués et peuvent alors être facilement calculées en temps polynomial. La définition de breakpoint donnée ci-dessus ne s'applique pas si G contient plusieurs répétitions d'un même gène. Cependant, supposer qu'un génome ne contient qu'une seule occurrence de chaque famille de gènes n'est pas valide biologiquement.

Une possibilité pour gérer ces duplications consiste à effectuer une *exemplarisation* des génomes, en ne gardant qu'un seul exemplaire de chaque famille de gènes (on retrouve donc une permutation). Parmi l'ensemble des choix possibles, l'exemplarisation garde le gène de chaque famille qui permet d'optimiser la mesure étudiée [San99] (voir Figure 8.6). Ceci est justifié par le fait que l'occurrence conservée du gène est supposée être le gène ancestral, depuis lequel les autres occurrences ont été obtenues.

Ceci mène au problème suivant :

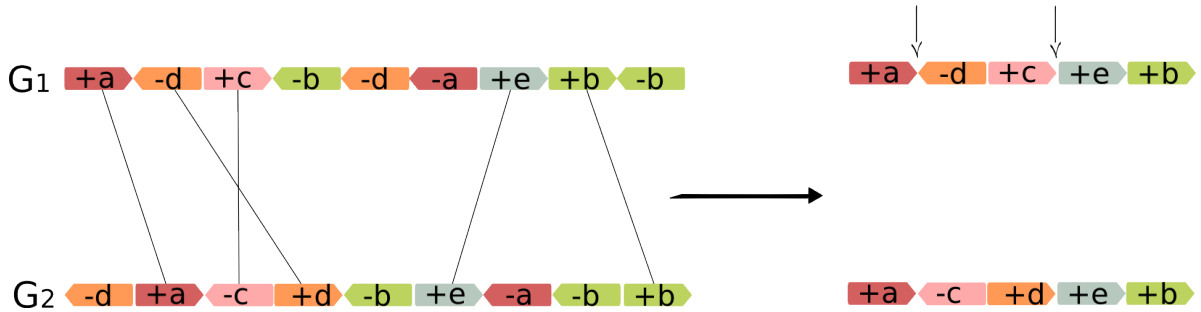


FIGURE 8.6 – Exemplarisation des deux génomes de gauche vers les deux génomes de droite, induisant deux breakpoints.

EXEMPLAR BREAKPOINT DISTANCE

- **Entrée** : Deux génomes G_1 et G_2 utilisant les mêmes gènes, un entier k.
- **Sortie** : Une exemplarisation de G_1 et G_2 pour laquelle le nombre de breakpoints est inférieur à k.

En considérant la version de minimisation de ce problème, s'il existait un algorithme d'approximation A de ratio r retournant une solution $A(I)$ pour une instance I, alors $A(I) \leq r \times OPT(I)$, où $OPT(I)$ est la solution optimale pour l'instance I. Si la solution optimale n'admet aucun breakpoint, alors $OPT(I) = 0$. Quelque soit le ratio r, $A(I)$ doit retourner 0 en temps polynomial. Si le problème EXEMPLAR BREAKPOINT DISTANCE où k est fixé à 0 est NP-Complet, alors un tel algorithme d'approximation est improbable, sauf si P $=$ NP. C'est pourquoi nous utilisons le cas particulier du problème EXEMPLAR BREAKPOINT DISTANCE suivant :

ZERO EXEMPLAR BREAKPOINT DISTANCE

- **Entrée** : Deux génomes G_1 et G_2 utilisant les mêmes gènes
- **Sortie** : Une exemplarisation de G_1 et G_2 pour laquelle le nombre de breakpoints est égal à 0.

Dans la suite, pour un génome G, on note $occ(G)$ le nombre maximal d'occurrences d'une famille de gènes dans G. Par exemple, dans la Figure 8.6, $occ(G_1) = 3$ car la famille du gène b apparaît trois fois. On note ZERO EXEMPLAR BREAKPOINT DISTANCE (p, q) le problème ZERO EXEMPLAR BREAKPOINT DISTANCE où $occ(G_1) = p$ et $occ(G_2) = q$.

8.2.2 Résultats connus

Les résultats concernant le problème EXEMPLAR BREAKPOINT DISTANCE sont donnés dans la Table 8.2. Bryant [Bry00] montre que le problème EXEMPLAR BREAKPOINT DISTANCE $(2, 1)$ est NP-Complet. Concernant les résultats sur son approximation, Angibaud *et al.* [AFR08] prouvent que le problème EXEMPLAR BREAKPOINT DISTANCE $(2, 1)$

q \ p	1	2	3
1	P	APX-Difficile [AFR08]	
2		?	?
3			Pas d'algo d'approx. [CFZ06]

TABLE 8.2 – Approximation du problème EXEMPLAR BREAKPOINT DISTANCE (p, q) avant notre article [BFSV09]. Les problèmes ouverts sont symbolisés par un point d'interrogation.

est APX-Difficile. De plus, Chen *et al.* [CFZ06] montrent qu'il n'existe pas d'algorithme d'approximation pour le problème EXEMPLAR BREAKPOINT DISTANCE $(3, 3)$. Cependant, ce résultat ne clôt pas totalement la question de l'approximation du problème EXEMPLAR BREAKPOINT DISTANCE. En effet, il est laissé comme problème ouvert la question de l'approximation du problème EXEMPLAR BREAKPOINT DISTANCE $(2, 2)$ [CFZ06, AFR$^+$09]. On note que pour résoudre le problème EXEMPLAR BREAKPOINT DISTANCE, les auteurs de [TNTZ05] proposent une heuristique tandis qu'Angibaud *et al.* [AFR$^+$07] proposent une méthode exacte basée sur de la programmation pseudo-booléenne.

Les résultats concernant le problème ZERO EXEMPLAR BREAKPOINT DISTANCE sont donnés dans la Table 8.3. On voit facilement que le problème ZERO EXEMPLAR BREAKPOINT DISTANCE $(p, 1)$ se résout en temps linéaire, pour $p \geq 1$. Chen *et al.* [CFZ06] montrent que le problème ZERO EXEMPLAR BREAKPOINT DISTANCE $(3, 3)$ est NP-Complet. Angibaud *et al.* [AFR$^+$09] montrent que le problème ZERO EXEMPLAR BREAKPOINT DISTANCE $(p, 2)$ est NP-Complet, mais avec une réduction où p est non borné. Par conséquent, la complexité du problème ZERO EXEMPLAR BREAKPOINT DISTANCE $(p, 2)$, où p est fixé reste ouverte.

q \ p	1	2	3	borné	non borné
1	P				
2		?	?	?	NP-Complet [AFR$^+$09]
3			NP-Complet [CFZ06]		

TABLE 8.3 – Complexité du problème ZERO EXEMPLAR BREAKPOINT DISTANCE (p, q) avant notre article [BFSV09]. Les problèmes ouverts sont symbolisés par un point d'interrogation.

Le résultat de la section suivante clôt les deux questions ouvertes évoquées précédemment.

8.2.3 Contributions

Dans [BFSV09], nous prouvons que ZERO EXEMPLAR BREAKPOINT DISTANCE $(2, 2)$ (et donc, ZERO EXEMPLAR BREAKPOINT DISTANCE $(p, 2)$ pour n'importe quel $p \geq 2$) est NP-Complet. Ce résultat fournit une caractérisation complète des cas polynomiaux et NP-Complets du problème, mais permet également de prouver que, sauf si P $=$ NP, il n'existe aucun algorithme d'approximation pour le problème EXEMPLAR BREAKPOINT DISTANCE, même lorsque chaque gène apparait au plus deux fois dans chaque génome.

Proposition 8.2.1. *Le problème* ZERO EXEMPLAR BREAKPOINT DISTANCE $(2, 2)$ *est* NP*-Complet.*

La réduction est effectuée depuis le problème 3-SAT. On remarque que Jiang a prouvé le même résultat plus tard, de manière indépendante [Jia10].

La question d'un algorithme d'approximation à ratio constant pour le problème EXEMPLAR BREAKPOINT DISTANCE $(p, 1)$, $p \geq 2$ reste ouverte. En effet, le problème ZERO EXEMPLAR BREAKPOINT DISTANCE $(p, 1)$ peut être résolu en temps polynomial et le problème EXEMPLAR BREAKPOINT DISTANCE $(p, 1)$ est APX-Difficile [AFR08].

On propose maintenant de contourner la difficulté du problème ZERO EXEMPLAR BREAKPOINT DISTANCE en étudiant sa complexité paramétrée, qui est également laissée comme question ouverte par [CFZ06].

En utilisant la technique du color-coding [AYZ95], on obtient le résultat suivant :

Proposition 8.2.2. *Le problème* ZERO EXEMPLAR BREAKPOINT DISTANCE *peut être résolu avec un algorithme ayant une complexité en temps de* $\mathcal{O}(m2^m)$*, où* m *est le nombre de familles de gènes.*

L'idée est d'assigner une couleur par famille de gènes, puis de construire un graphe orienté $D = (V, A)$ où il existe un sommet pour chaque paire de gènes de la même famille portant le même signe. Étant donnés deux sommets $\{i, j\}$ et $\{p, q\}$ de V, il existe une arête entre $\{i, j\}$ et $\{p, q\}$ si $i < p$ et $j < q$. On peut ensuite prouver qu'il existe un chemin colorful dans D si et seulement si il existe une exemplarisation de G_1 et G_2 sans breakpoints (voir Figure 8.7).

Ce nombre m étant potentiellement proche de la taille du génome, on propose un second algorithme de complexité paramétrée dépendant du *span*, la distance maximale entre deux occurrences d'un même gène.

Proposition 8.2.3. *Le problème* ZERO EXEMPLAR BREAKPOINT DISTANCE *peut être résolu avec un algorithme ayant une complexité en temps de* $\mathcal{O}(n2^{2s}s^3)$*, où* n *est la longueur du plus court génome et* s *est le span maximum.*

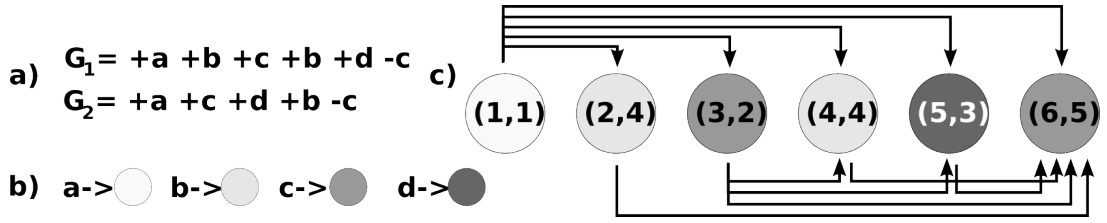


FIGURE 8.7 – a) Deux génomes G_1 et G_2 ; b) Une fonction de coloration ; c) Le graphe orienté $D = (V, A)$ correspondant.

8.3 Intervalles communs de génomes

Ce travail est effectué en collaboration avec Xiao Yang, Guillaume Blin, Sylvie Hamel, Romeo Rizzi et Srinivas Aluru [YSB+11].

8.3.1 Motivations

Comme précisé dans la section précédente, durant l'évolution, certaines sous-séquences d'ADN divergent, tandis que d'autres sont conservées parmi différents organismes. Beaucoup de ces sous-séquences conservées correspondent à des éléments fonctionnels (des gènes), qui sont particulièrement important pour comprendre l'évolution. Par conséquent, dans de nombreuses études, des événements tels que l'insertion de gènes, la perte de gènes, la duplication de gènes ou encore l'inversion de gènes ont un rôle dans l'évolution. Deux gènes ayant des séquences très similaires, typiquement résultant d'une spéciation ou d'une duplication, sont considérés comme appartenant à une même famille de gènes. Dans la suite, une famille de gènes et les gènes la constituant sont étiquetés de la même manière.

Un tâche importante dans la génomique comparative consiste à identifier un ensemble de gènes ayant des emplacements proches. En effet, la proximité d'un ensemble de gènes tend à indiquer que les gènes correspondants forment une unité fonctionnelle (appelée dans la littérature *gene cluster*), ou résultent d'un événement de spéciation ou de duplication [HD05] (connu sous le nom de *synteny*). Ces deux cas ont été intensivement étudiés ces dernières années et plusieurs modèles et algorithmes ont été proposés pour les définir et les identifier – nous en donnons un panorama dans la section suivante.

Dans la section précédente, étant donnés deux génomes, nous cherchions à optimiser une certaine distance. Cette fois-ci, de manière informelle, étant donnés un ensemble A de gènes dits ancestraux et un ensemble de chromosomes, on recherche les régions (sous-séquences) de ces chromosomes qui correspondent à A. Les critères biologiques nécessaires pour décider si une région est d'intérêt sont connus difficiles à déterminer. Cependant, quelques propriétés essentielles ont été soulevées par Hoberman et Durand [HD05] :

– un gène est considéré comme ancestral s'il apparaît dans un nombre minimum de

chromosomes,
- une région (sous-séquence) d'un chromosome considérée comme d'intérêt doit contribuer suffisamment à l'ensemble de gènes ancestraux et doit donc en contenir un nombre minimum,
- chaque région considérée doit avoir une certaine densité locale de gènes ancestraux, c'est-à-dire un nombre limité d'insertions de gènes entre deux gènes ancestraux,
- l'ensemble des régions considérées doit avoir une certaine densité globale, c'est-à-dire un nombre limité de délétions de gènes sur chaque région et un nombre global de délétions de gènes limité.

8.3.2 Résultats connus

Étant donnés $|S|$ chromosomes représentés comme des permutations sur un ensemble de gènes, un CONSERVED SEGMENT [NT84] correspond à un ensemble de gènes qui apparaissent de manière consécutive dans le même ordre sur chaque chromosome.

Si la contrainte demandant que les gènes apparaissent de manière consécutive est supprimée, on obtient la définition du modèle COMMON INTERVAL [UY00]. Si le premier et le dernier gène d'un COMMON INTERVAL sont identiques sur les k chromosomes, on parle alors de CONVERVED COMMON INTERVAL [BS06].

Si la contrainte demandant que les gènes d'un COMMON INTERVAL doivent être consécutifs dans chaque chromosome est supprimée (donc deux gènes d'une solution peuvent avoir un nombre borné de gènes n'appartenant pas à la solution entre eux), on obtient la définition du modèle GENE TEAMS, modèle également appelé MAX-GAP [BCR02]. Ce modèle est plus pertinent biologiquement puisqu'il autorise des insertions de gènes.

Plus récemment, le modèle APPROXIMATE COMMON INTERVAL a été introduit [RK06, BJMS09]. Contrairement aux modèles précédents, on ne demande pas que tous les gènes d'intérêt soient présents dans chaque occurrence de la solution.

Nous présentons maintenant le modèle appelé MULTI RELATED SEGMENTS, que nous avons introduit dans [YSB+11] et qui tente d'unifier l'ensemble des notions précédentes. Chaque sous-séquence de la solution est considérée comme descendante d'un ensemble de gènes ancestraux, avec des insertions, des pertes, des duplications et des inversions de gènes. Définissons formellement un MULTI RELATED SEGMENT. Pour assurer que ce sont bien des gènes ancestraux, chaque gène de l'ensemble A de gènes ancestraux doit apparaître dans β sous-séquences (régions) des chromosomes S. Chaque sous-séquence de la solution doit contenir au moins ϵ_m gènes ancestraux différents, et doit être maximale (on ne doit pas pouvoir l'agrandir en ajoutant des gènes voisins) – ainsi, chaque sous-séquence doit contribuer suffisamment à A. Comme dans le modèle GT, on autorise au plus α gènes non ancestraux entre deux gènes ancestraux

d'une sous-séquence donnée, ceci afin d'obtenir une densité locale de gènes ancestraux. Enfin, pour obtenir une densité globale de gènes ancestraux, on demande que chaque sous-séquence ne subisse pas plus de ϵ_l pertes de gènes, et que le nombre total de gènes perdus soit inférieur à ϵ_t. Finalement, étant donnés un ensemble de gènes ancestraux A, un ensemble de chromosomes et les paramètres $\alpha, \beta, \epsilon_m, \epsilon_l$ et ϵ_t, le problème consiste à identifier tous les MULTI RELATED SEGMENTS (voir Figure 8.8).

$$A = \{a, b, c, d\}$$

$$S_1 = ...e\ \mathbf{b\ f\ a\ c}\ e...$$
$$S_2 = ...\mathbf{d\ a\ e\ c\ b}\ f...$$
$$S_3 = ...\mathbf{b\ c\ d}\ f\ e\ \mathbf{a\ d\ b}...$$

FIGURE 8.8 – Exemple où l'ensemble des gènes considéré est $\{a, b, \ldots, f\}$, l'ensemble de gènes ancestraux recherché est $A = \{a, b, c, d\}$, l'ensemble des chromosomes $S = \{S_1, S_2, S_3\}$ est formé de séquences sur l'ensemble de gènes. Les différents MULTI RELATED SEGMENTS sont identifiés en gras. Ces sous-séquences respectent les paramètres $\beta = 2$ (chaque gène ancestral apparaît dans au moins 3 sous-séquences), $\epsilon_m = 3$ (chaque sous-séquence contient au moins 3 gènes ancestraux), $\alpha = 1$ (il y a au plus un gène non-ancestral entre deux gènes ancestraux), $\epsilon_l = 1$ (il y a au plus une perte de gène par sous-séquence) et $\epsilon_t = 3$ (il y a au total trois pertes de gènes).

Par rapport aux modèles existants, la définition de MULTI RELATED SEGMENTS a plusieurs avantages. Premièrement, on retrouve les modèles précédents. En effet, en plaçant $\beta = |S|$, $\epsilon_m = |A|$ et $\alpha = 0$ (où S est l'ensemble des chromosomes d'entrée et A est l'ensemble de gènes ancestraux), on obtient un COMMON INTERVAL, tandis qu'en plaçant $\alpha \geq 0$, on obtient un GENE TEAM. Par rapport à ces deux modèles, MULTI RELATED SEGMENT gère en plus les événements de pertes de gènes. Cet aspect est déjà considéré par le modèle APPROXIMATE COMMON INTERVAL, cependant, entre autres différences, la densité locale de gènes ancestraux n'est pas prise en compte dans ce dernier modèle, ce qui est pourtant crucial d'après [HD05].

D'un point de vue algorithmique, la complexité des modèles présentés augmente significativement lorsque des séquences sont considérées au lieu de permutations, ce qui est plus proche de la réalité biologique, où il peut exister plusieurs occurrences d'un même gène. Le problème est polynomial si l'on considère les COMMON INTERVAL [BCMR08], mais devient difficile si l'on considère les CONSERVED COMMON INTERVAL [BCF+07]. Le modèle GENE TEAM est polynomial sur les permutations [BCR02], mais exponentiel sur les séquences [PBR+05]. La complexité du modèle APPROXIMATE COMMON INTERVAL reste quant à elle ouverte [RK06, BJMS09].

Dans la suite de cette section, on énonce des résultats algorithmiques concernant le problème MULTI RELATED SEGMENT, selon différentes restrictions sur le modèle, permettant de donner les premiers résultats de complexité lorsque des pertes de gènes sont autorisées.

8.3.3 Contributions

Nous avons dans un premier temps étudié le cas où l'ensemble A de gènes ancestraux est connu *a priori*. Le problème, appelé LOCATEMRS, consiste donc à chercher sur $|S|$ chromosomes représentés par des séquences, tous les MULTI RELATED SEGMENTS correspondant à A.

Malheureusement, le problème est NP-Complet, même dans un cas plus restreint.

Proposition 8.3.1. *Le problème* LOCATEMRS *est* NP*-Complet, même lorsque chaque chromosome est représenté par une permutation et au plus une sous-séquence de chaque chromosome peut être dans la solution.*

Cette difficulté est cependant contre-balancée par le fait que ce même problème peut être résolu au moyen d'un algorithme de complexité paramétrée (avec pour paramètre le nombre de gènes ancestraux) se basant sur de la programmation dynamique.

Proposition 8.3.2. *On peut résoudre le problème* LOCATEMRS *à l'aide d'un algorithme de complexité* $\mathcal{O}((|S|3^{|A|})^2)$.

Dans un second temps, on considère le problème MAXMRS, où l'ensemble des gènes ancestraux A n'est pas connu. Le but est alors de maximiser la taille de A. Malheureusement, une fois encore, le problème est difficile à approximer, répondant à une question ouverte de [YA10]

Proposition 8.3.3. *Le problème* MAXMRS *est* APX*-Difficile, même lorsque chaque chromosome est représenté par une permutation.*

Cependant, si ni le nombre de gènes perdus par sous-séquence d'intérêt, ni le nombre de sous-séquences d'intérêt par chromosome ne sont restreints, alors on peut résoudre le problème MAXMRS avec un algorithme polynomial.

Proposition 8.3.4. *On peut résoudre le problème* MAXMRS *avec un algorithme polynomial si ni le nombre de gènes perdus par sous-séquences d'intérêt, ni le nombre de sous-séquences d'intérêt par chromosomes ne sont restreints.*

La complexité du problème MAXMRS reste ouverte lorsque le nombre de sous-séquences d'intérêt par chromosome est non borné et que le nombre maximum de gènes perdus par sous-séquence d'intérêt est borné.

8.4 Minimiser les duplications de gènes

La suite de cette section est le fruit d'une collaboration avec Guillaume Blin, Paola Bonizzoni, Riccardo Dondi et Romeo Rizzi [BBD+12].

8.4.1 Motivations et définitions

Comme déjà évoqué, les génomes des organismes actuels sont le fruit d'une série d'événements, parmi lesquelles on trouve la *spéciation*, mécanisme caractérisé par l'apparition de nouvelles espèces depuis un ancêtre commun. Retrouver les différents événements menant aux espèces actuelles est un problème largement étudié en génomique comparative. Cet historique est souvent représenté par un type spécial d'arbre phylogénétique, appelé *arbre des espèces* (*species tree*). L'arbre des espèces est considéré comme un arbre binaire enraciné, où chaque espèce existante d'un ensemble Λ (représentant le génome de chaque espèce) est l'étiquette d'une seule feuille de l'arbre, et où l'ancêtre commun des espèces contemporaines est associé à la racine de l'arbre. Les sommets internes représentent des espèces ancestrales (et les spéciations associées).

Généralement, pour construire un arbre des espèces, on construit l'arbre phylogénétique de chaque gène relevé dans ces espèces. On suppose donc que l'évolution de ces gènes imite l'évolution des espèces elles-mêmes. Étant donnée une famille de gènes, on appelle *gene tree* l'arbre binaire représentant les relations d'évolution de cette famille de gènes à travers l'ensemble Λ des espèces. Par conséquent, les sommets du *gene tree* représentent des gènes, comme étant les différentes versions de la famille de gènes considérée. Il existe de plus une affection (*sample-mapping*) entre chaque feuille d'un *gene tree* et les espèces se situant sur les feuilles de l'arbre des espèces. Cette affectation donne une étiquette représentant l'espèce où le gène a été relevé. Par exemple, sur la Figure 8.9, le gène a a été relevé dans le génome de l'espèce A. Plusieurs feuilles du *gene tree* peuvent avoir la même étiquette, représentant deux gènes apparaissant dans une même espèce. Si chaque feuille du *gene tree* peut être associé à une feuille de l'arbre des espèces, les deux arbres sont dits *comparables*.

Cependant, suite à des processus complexes de l'évolution déjà évoqués tels que la perte ou la duplication de gènes, les recombinaisons ou les transferts horizontaux, l'arbre représentant l'évolution de chaque gène ne correspond pas toujours à l'évolution réelle des espèces correspondantes. Dans la suite de cette section, on se concentre sur la duplication de gènes qui est un événement connu essentiel dans l'évolution d'espèces telles que les vertébrés, les insectes ou les plantes [ES03]. Un problème largement étudié consiste à réconcilier un *gene tree* avec l'arbre des espèces en considérant d'hypothétiques duplications de gènes. Par exemple, Figure 8.9, étant donnés un *gene tree* et un arbre des espèces comparables induisant des incompatibilités, on peut construire un arbre de réconciliation basé sur la duplication *a priori* d'un gène x vers deux copies x' et x'', qui ont par la suite subi des spéciations selon la topologie de l'arbre des espèces. Le *gene*

tree est réconcilié par la duplication du gène x, permettant d'expliquer l'incompatibilité.

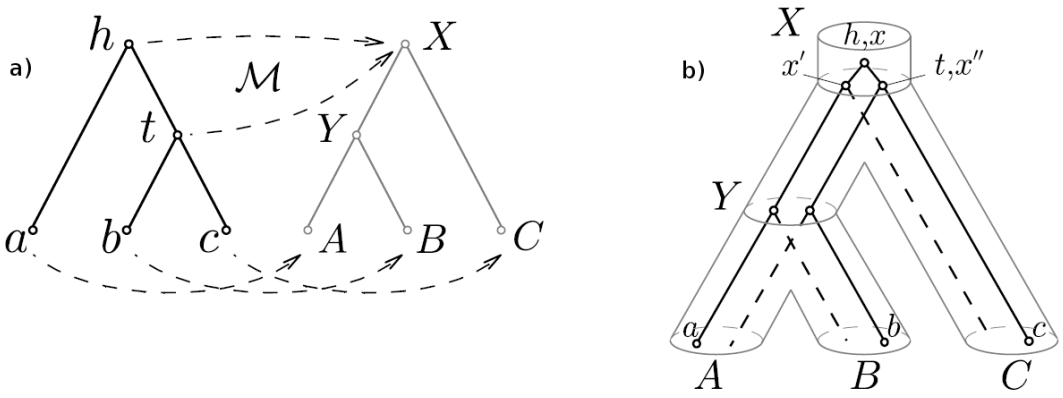


FIGURE 8.9 – a) Un *gene tree* et un arbre des espèces. L'affectation $\mathcal{M}$ associe à chaque gène g l'ancêtre commun le plus proche où tous les descendants de g ont été relevés (par exemple, le gène t a été relevé dans B et C, dont X est l'ancêtre commun le plus proche). Chaque feuille du *gene tree* est associée à une feuille de l'arbre des espèces où le gène a été relevé – les deux arbres sont compatibles. b) Un arbre de réconciliation basé sur la duplication du gène x relevé dans l'espèce X en deux gènes x' et x''. Les lignes pleines représentent le *gene tree* dans l'arbre de réconciliation. La figure provient de [BBEW07].

Selon le principe de parcimonie, on cherche à trouver le nombre minimum de gènes dupliqués pour résoudre toutes les incompatibilités [GCM+79]. Ceci peut-être effectué par l'*affectation de l'ancêtre commun le plus proche* (*LCA-mapping*), noté $\mathcal{M}$ et défini comme suit. On affecte par $\mathcal{M}$ tous les gènes du *gene tree* à la dernière espèce où le gène a pu être relevé. En d'autres termes, $\mathcal{M}$ affecte chaque gène ancestral g du *gene tree* à l'ancêtre commun le plus récent des espèces existantes, d'où tous les descendants de g ont été relevés. Par exemple, selon l'affectation $\mathcal{M}$ de la Figure 8.9, le gène t est affecté à X car X est l'ancêtre commun le plus récent de B et C, où ont été relevés respectivement les descendant b et c de t. Selon $\mathcal{M}$, un gène du *gene tree* est dit dupliqué s'il a un descendant ayant la même affectation par $\mathcal{M}$. Par exemple, Figure 8.9, le gène h et son fils t sont assignés à la même espèce X. Alors, le coût de la réconciliation est défini par le nombre de duplications de gènes dans le *gene tree* induit par l'arbre des espèces. Le calcul de cette distance a été largement étudié dans le contexte du problème MINIMUM DUPLICATION.

MINIMUM DUPLICATION

- **Entrée** : Un ensemble de *gene trees*.
- **Sortie** : Un arbre des espèces compatible.
- **Mesure** : Le nombre de duplications de gènes.

Récemment, le problème MINIMUM DUPLICATION BIPARTITE a été introduit pour pouvoir attaquer le problème MINIMUM DUPLICATION [OSC10].

MINIMUM DUPLICATION BIPARTITE

- **Entrée** : Un ensemble de gènes G, un ensemble de *gene trees* sur G.
- **Sortie** : Une bipartition de G.
- **Mesure** : Le nombre de pré-duplications.

Ce problème consiste à minimiser les *pré-duplications*, c'est-à-dire les duplications qui, au cours de l'évolution, précèdent la première spéciation selon l'arbre des espèces. Autrement dit, seul le premier niveau de l'arbre des espèces est considéré. En effet, on s'intéresse à savoir si une certaine espèce appartient au sous-arbre de l'arbre des espèces enraciné au fils gauche de la racine, ou bien au fils droit. Par conséquent, on peut considérer l'arbre des espèces comme étant une bipartition (Λ_1, Λ_2) de l'ensemble des espèces Λ (voir Figure 8.10). Résoudre récursivement le problème MINIMUM DUPLICATION BIPARTITE produit une heuristique gloutonne naturelle pour le problème MINIMUM DUPLICATION.

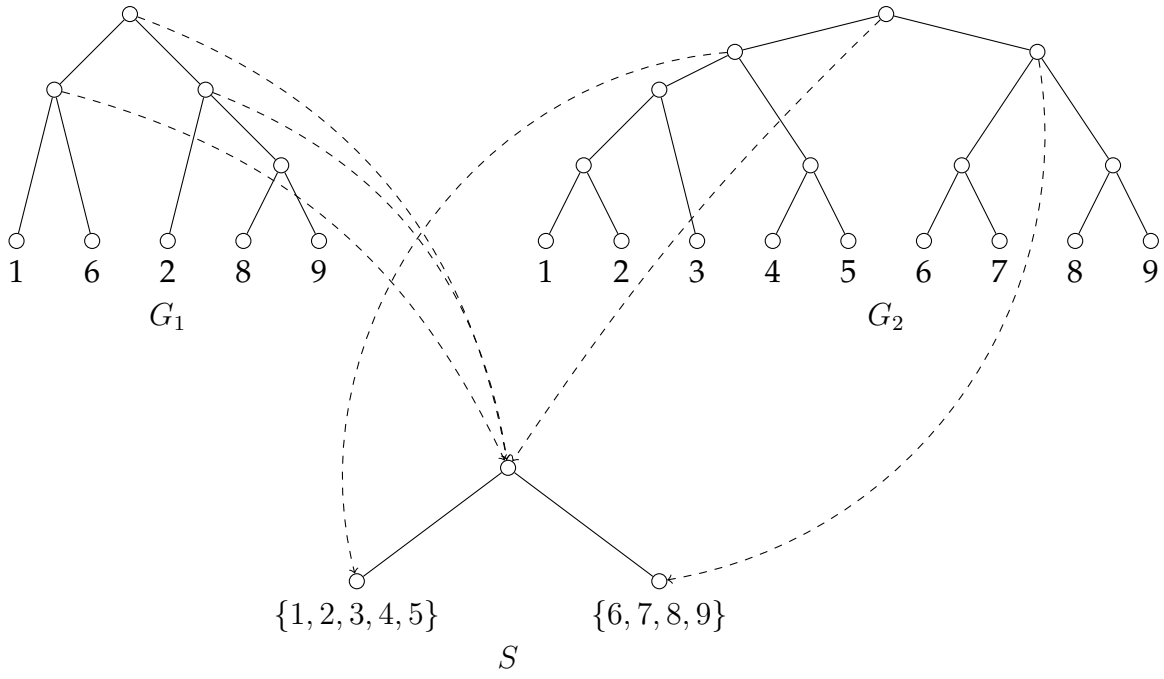


FIGURE 8.10 – Deux *gene trees* G_1 et G_2, et l'arbre des espèces S qui est une bipartition $(\{1,2,3,4,5\},\{6,7,8,9\})$. Pour plus de lisibilité, nous avons étiqueté les feuilles de G_1 et G_2 directement avec les espèces de S. Le seul sommet subissant une pré-duplication est la racine de G_1, puisque lui et ses deux fils sont assignés par $\mathcal{M}$ (représentée par les flèches en pointillés) à la racine de S. Toujours pour plus de lisibilité, nous n'avons pas dessiné l'affectation $\mathcal{M}$ pour chaque sommet de G_1 et G_2.

8.4.2 Résultats connus

Le problème MINIMUM DUPLICATION est NP-Difficile [MLZ00]. Plus récemment, il a été mis en relation avec le problème MINIMUM ROOTED TRIPLETS CONSISTENCY [BS11].

La complexité du problème MINIMUM ROOTED TRIPLETS CONSISTENCY a été largement étudiée, et ce problème est non seulement NP-Difficile, mais aussi W[2]-Difficile [BGJ10] et il n'existe pas pour ce problème d'algorithme d'approximation ayant un ratio inférieur à $\mathcal{O}(\log n)$, sauf si P = NP, où n est le nombre total de feuilles étiquetées différemment en entrée [BGJ10].

Ces résultats, couplés à la réduction donnée dans [BS11], impliquent que le problème MINIMUM DUPLICATION est NP-Difficile, W[2]-Difficile (ce qui contredit le résultat de complexité paramétrée donné dans [Ste99], montré faux par [BS11]). De plus, il n'existe pas pour ce problème d'algorithme d'approximation avec un ratio inférieur à $\mathcal{O}(\log n)$ sauf si P = NP, même dans le cas spécifique d'une forêt constituée d'un nombre non borné d'arbres d'un gène à trois feuilles où chaque étiquette est unique [BS11]. Par conséquent, différentes heuristiques ont été développées [BBEW07, BEW09], ainsi qu'une méthode exacte basée sur une formulation de programmation entière [CBFBE11].

Concernant le problème MINIMUM DUPLICATION BIPARTITE, nous avons déjà précisé que le résoudre récursivement produit une heuristique gloutonne pour le problème MINIMUM DUPLICATION. Un algorithme de 2-approximation a été donné pour le problème MINIMUM DUPLICATION BIPARTITE [OSC10], mais sa complexité reste ouverte.

8.4.3 Contributions

Dans [BBD+12], nous avons contribué de deux manières aux deux problèmes précédemment définis. Premièrement, nous avons étudié l'approximation du problème MINIMUM DUPLICATION quand le nombre d'arbres donnés dans l'entrée du problème est borné (on sait déjà qu'il n'existe pas d'algorithme d'approximation ayant un ratio inférieur à $\mathcal{O}(\log n)$ si le nombre d'arbres est non borné, sauf si P = NP).

Proposition 8.4.1. *Le problème* MINIMUM DUPLICATION *est* APX*-Difficile, même lorsque seulement cinq gene trees sont considérés en entrée et chaque étiquette n'est attribuée qu'une seule fois.*

La réduction s'effectue depuis le cas particulier du problème VERTEX COVER quand le graphe d'entrée est cubique (c'est-à-dire que tous les sommets ont un degré égal à trois).

Dans un second temps, on montre qu'il est possible de résoudre le problème MINIMUM DUPLICATION BIPARTITE par un algorithme polynomial randomisé si les *gene trees* en entrée ont une hauteur bornée. Plus précisément, il existe une procédure polynomiale dont on connait la probabilité d'erreur. En relançant un nombre polynomial de fois cette procédure, on diminue autant que voulu la marge d'erreur.

Proposition 8.4.2. *On peut résoudre le problème* MINIMUM DUPLICATION BIPARTITE *en temps polynomial à l'aide d'un algorithme randomisé si la hauteur des arbres d'un gène en entrée est bornée.*

L'idée de cet algorithme est d'adapter un algorithme polynomial randomisé connu (voir [KT05] pour une description détaillée) permettant de résoudre le problème MINIMUM CUT.

MINIMUM CUT

- **Entrée** : Un graphe $G = (V, E)$.
- **Sortie** : Une partition de V en deux ensembles disjoints non vides A et B.
- **Mesure** : Le nombre d'arêtes $\{u, v\} \in E$ telles que $u \in A$ et $v \in B$.

Nous adaptons l'algorithme pour qu'il soit valide sur la variante considérant des hypergraphes colorés, où les hyper-arêtes ont un degré borné, ce qui est, à notre connaissance la première approche randomisée pour répondre à la complexité de ce problème. Précisons que nous ne nous servons pas de l'algorithme pour le problème MINIMUM CUT en tant que sous-routine (un algorithme polynomial déterministe existe pour ce problème), mais que nous adaptons l'idée utilisée dans l'algorithme randomisé pour le problème MINIMUM CUT.

La complexité du problème MINIMUM DUPLICATION BIPARTITE pour des arbres de hauteur non bornée reste un problème ouvert. Nous pensons qu'une approche possible consiste à déterminer la complexité du problème MINIMUM CUT sur les hypergraphes colorés ayant un degré non borné.

Chapitre 9

Comparaison de structures secondaires d'ARN

Contenu

Ce court chapitre traite d'un travail concernant la comparaison de deux structures secondaires d'ARN. Il est le fruit d'une collaboration avec Guillaume Blin, Romeo Rizzi et Stéphane Vialette. Étant toujours en cours d'étude, ce travail n'est pas publié.

9.1 Motivations et définitions

En génomique comparative, il est souvent question de comparaison de séquences. Dans le contexte de l'étude de l'ARN, nous avons vu Section 2.1.2 qu'on ne pouvait pas uniquement utiliser la séquence de nucléotides, car la structure de la molécule joue un rôle important dans la fonction portée par l'ARN. Par conséquent, la suite de cette section porte sur la structure secondaire de l'ARN (voir Section 2.1.2 pour la définition des structures de l'ARN).

Ces dernières années, en raison de l'augmentation du nombre de structures d'ARN disponibles, il est devenu crucial de pouvoir les comparer. D'un point de vue théorique, la comparaison de structures secondaires d'ARN a été étudiée selon différents paradigmes, autorisant diverses formes de flexibilités sur le critère de comparaison [Eva99, JLMZ04, AGGN04]. Malheureusement, pour la plupart de ces paradigmes, la comparaison de telles structures mène à des problèmes difficiles algorithmiquement. Cependant, quelques articles ont montré que relaxer les contraintes sur la pré-

servation de la structure primaire rend le problème polynomial pour certains cas [LV04, KRVW06, Eva06, MWB09].

D'un point de vue combinatoire, on peut distinguer deux types de modèles d'encodage de structures d'ARN autorisant diverses formes de flexibilité :

- Une représentation qui inclut la séquence de nucléotides ainsi que les liaisons hydrogènes formant la structure, appelée *séquence arc-annotée* [Eva99].
- Une représentation utilisant des graphes, ne prenant pas nécessairement en compte les nucléotides composant la séquence. De tels graphes peuvent être les graphes de 2-intervalles [Via04] ou les graphes linéaires [DB06, Eva06].

Dans la suite de cette section, nous nous intéressons particulièrement à des graphes linéaires arêtes-disjoints, c'est-à-dire un graphe avec un ordre linéaire des sommets, qui sont incidents à exactement une arête. Autrement dit, tous les sommets du graphe peuvent être placés sur une ligne droite et deux arêtes ne partagent pas de sommets communs (c'est un couplage). Les graphes linéaires peuvent représenter une structure d'ARN où l'identification des nucléotides n'est pas prise en compte. En effet, l'ordre de la séquence d'ARN est préservée grâce à l'ordre linéaire des sommets et les arêtes représentent les liaisons hydrogènes. Étant données deux arêtes $\{i, j\}$ et $\{h, l\}$, où $i < j$ et $h < l$, on dit généralement qu'elles ont une relation (voir aussi Figure 9.1) :

- de *précédence* si $j < h$ ou $l < i$,
- d'*inclusion* si $i < h < l < j$ ou $h < i < j < l$,
- de *croisement* si $i < h < j < l$ ou $h < i < l < j$.

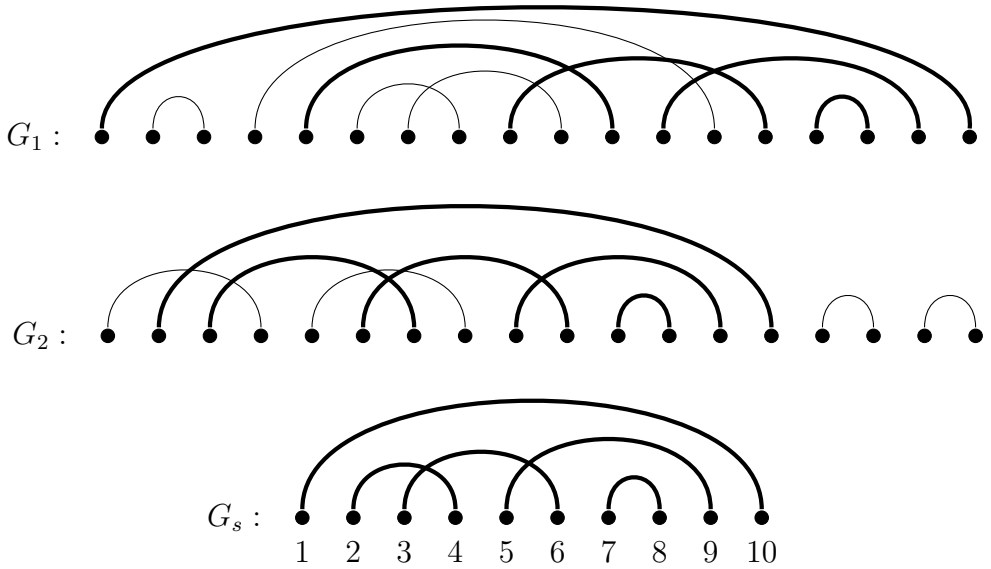


FIGURE 9.1 – Deux graphes linéaires (ici en plus arêtes-disjoints) G_1 et G_2 et leur sous-graphe de taille maximum G_s (également reporté en gras dans G_1 et G_2). Les arêtes $\{2, 4\}, \{3, 6\}, \{5, 9\}$ et $\{7, 8\}$ sont inclues dans l'arête $\{1, 10\}$. Les arêtes $\{2, 4\}$ et $\{3, 6\}$ précèdent l'arête $\{7, 8\}$. L'arête $\{2, 4\}$ précède l'arête $\{5, 9\}$ et croise l'arête $\{3, 6\}$, tandis que l'arête $\{3, 6\}$ croise les deux arêtes $\{2, 4\}$ et $\{5, 9\}$.

Pour comparer deux graphes linéaires, on dit qu'un graphe G_1 est un sous-graphe de G_2 si on peut obtenir G_1 depuis G_2 par une séquence de suppression d'arêtes et de sommets.

Le problème (de décision) concernant la comparaison de structures d'ARN représentées par des graphes linéaires est alors défini comme suit (et appelé MCOS dans [Eva06]) :

MAXIMUM COMMON LINEAR GRAPH

- **Entrée** : Deux graphes linéaires G_1 et G_2, un entier k.
- **Sortie** : Un sous-graphe linéaire commun de taille supérieure à k.

Voir la Figure 9.1 pour un exemple. Ce problème revient à trouver la structure secondaire ayant au moins k sommets, commune aux deux structures d'entrée. Précisons que l'ordre sur les sommets est important et doit être conservé.

9.2 Résultats connus

Concernant le problème MAXIMUM COMMON LINEAR GRAPH, si au moins l'un des deux graphes d'entrée est arêtes-disjoints et ne contient aucune arête se croisant, le problème se résout avec un algorithme polynomial ayant une complexité de $O(n^4 \log^3 n)$, où n est le plus grand nombre de sommets des deux graphes d'entrée [KRVW06]. Précédemment, Lozano and Valiente [LV04] ont proposé un algorithme polynomial ayant une complexité de $O(n^2m^2)$ (où m et n représentent respectivement le nombre de sommets dans G_1 et G_2) si les deux graphes ont les deux restrictions précédentes.

Malheureusement, le problème devient NP-Complet si l'une des deux restrictions n'est pas respectée dans les deux graphes d'entrée [BBL98, Via04]. C'est particulièrement un problème car, comme précisé par Staple et Butcher dans [SB05], les pseudo-nœuds (qui provoquent des croisements d'arêtes) sont relativement fréquents et jouent un rôle important. Heureusement, comme montré par Evans [Eva06], il est possible d'espérer des algorithmes polynomiaux si on ne considère que certaines classes de pseudo-nœuds. En effet, les résultats de difficultés précédents utilisent des constructions où le nombre d'arêtes se croisant est arbitraire et ne ressemble pas aux structures d'ARN connues. En réalité, la plupart des structures d'ARN sont *plongées en 2-pages*, c'est-à-dire séparables en deux demi-plans tels que les arêtes de chaque demi-plan (ou chaque page) ne se croisent pas. Par exemple, on peut plonger en 2-pages le graphe linéaire de la Figure 9.2 en plaçant les trois premières arêtes ($\{1, 9\}, \{2, 7\}$ et $\{3, 5\}$) dans le demi-plan inférieur.

Le problème reste difficile, même pour les structures plongées en 2-pages [GIP99], mais avec la condition supplémentaire que les arêtes de chaque structure ne soient pas indépendantes, c'est-à-dire que des sommets peuvent être partagés par des arêtes de

pages différentes. Evans [Eva06] propose un algorithme polynomial ayant une complexité de $\mathcal{O}(n^{10})$ (où n est le plus grand nombre de sommets entre les deux structures) pour résoudre le problème MAXIMUM COMMON LINEAR GRAPH, mais dans le cas spécifique où les graphes sont arêtes-disjoints, les structures sont plongées en deux pages et les *interleaved left-right endpoints* sont interdits. Étant donné un graphe linéaire G_1 représentant une structure d'ARN, un *interleaved left-right endpoints* correspond à un sous-graphe $G' = \{(1,9),(2,7),(3,5),(4,12),(6,11),(8,10)\}$ apparaissant dans G_1 (voir aussi Figure 9.2). Si la complexité pire cas semble rendre impossible une implémentation, l'auteur précise qu'en pratique, l'algorithme s'exécute « convenablement » (il n'y a pas de précision sur le temps d'exécution) [Eva06]. Nous pouvons supposer que les cas induisant une hausse de la complexité ne sont pas courant sur des données réelles.

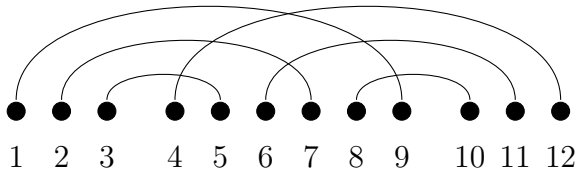


FIGURE 9.2 – Structure d'un *interleaved left-right endpoints*

La complexité du problème MAXIMUM COMMON LINEAR GRAPH pour des graphes linéaires arêtes-disjoints et plongés en 2-pages – noté 2-MAXIMUM COMMON LINEAR GRAPH – est laissée comme problème ouvert par Evans [Eva06].

9.3 Contributions

Nous montrons la difficulté du problème 2-MAXIMUM COMMON LINEAR GRAPH *via* une réduction depuis une variante du problème NP-Complet 3-DIMENSIONAL MATCHING [GJ79].

3-DIMENSIONAL MATCHING

• **Entrée** : Un entier q, trois ensembles distincts X, Y, Z, contenant tous q éléments, un ensemble T de triplets (x, y, z) de $X \times Y \times Z$ tel que $x \in X, y \in Y, z \in Z$.

• **Sortie** : Un sous-ensemble $T' \subseteq T$ tel que (i) $|T'| = q$ et (ii) étant donnés $(x_i, y_i, z_i) \in T'$ et $(x_j, y_j, z_j) \in T'$, $x_i \neq x_j, y_i \neq y_j$ et $z_i \neq z_j$?

Depuis n'importe quelle instance $\mathcal{I} = (T, X, Y, Z)$ du problème 3-DIMENSIONAL MATCHING, on peut lui associer un graphe biparti (voir Figure 9.3). La bipartition du graphe se trouve entre les sommets représentant les ensembles X, Y et Z, et les sommets représentant les triplets de T. Pour chaque triplet t, on définit une arête entre le sommet représentant t et les sommets représentant les éléments de t.

Une étude préliminaire montre que s'il est possible de déterminer avec un algorithme de complexité paramétrée si un graphe linéaire arêtes-disjoints apparaît entièrement dans un autre graphe linéaire arêtes-disjoints, alors le problème 2-MAXIMUM COMMON LINEAR GRAPH est dans la classe FPT avec pour paramètre k, le nombre de sommets de la solution.
Pour cela, on génère l'ensemble des graphes linéaires arêtes-disjoints de taille k pouvant être des solutions potentielles, en remarquant que ces graphes sont équivalents aux manières de mélanger un mot bien parenthésé de taille k' (correspondant à la page supérieure) avec un mot bien parenthésé de taille $k - k'$ (correspondant à la page inférieure). Chaque parenthèse représente un sommet et une parenthèse fermante forme un arc avec la dernière parenthèse ouverte non encore fermée. Cet ensemble de graphes est de taille exponentielle selon k uniquement. Par la suite, il faut déterminer si chaque graphe linéaire de taille k apparaît entièrement dans les deux graphes linéaires arêtes-disjoints de l'entrée.

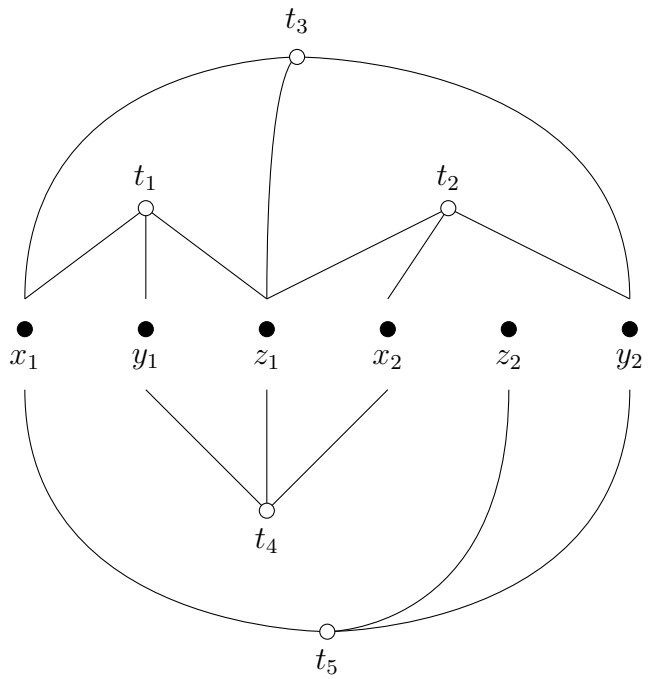


FIGURE 9.3 – Graphe planaire biparti représentant l'instance du problème PLANAR 3DM où $T = (x_1, y_1, a_1), (x_2, y_2, z_1), (x_1, y_2, z_1), (x_2, y_1, z_1), (x_1, y_2, z_2)$.

Dyer et Frieze ont prouvé que le problème 3-DIMENSIONAL MATCHING reste NP-Complet même pour la classe des instances où le graphe biparti associé est planaire et connexe [DF86]. Le problème reste de plus NP-Complet s'il existe un cycle entre les sommets de X, Y et Z.

L'idée de notre construction consiste à considérer les sommets représentant les éléments de X, Y et Z de l'instance du problème 3-DIMENSIONAL MATCHING comme

les sommets du graphe linéaire pour l'instance de 2-MAXIMUM COMMON LINEAR GRAPH. Les sommets des triplets sont donc sur chaque page. Comme le graphe biparti est planaire, on peut facilement trouver une construction où les arêtes ne se croisent pas. Ceci nous mène à la proposition suivante.

Proposition 9.3.1. *Le problème 2-*MAXIMUM COMMON LINEAR GRAPH *est* NP*-Complet.*

Conclusion et perspectives

Cette thèse a permis l'étude algorithmique de différents problèmes concernant les réseaux biologiques, mais aussi celle de problèmes classiques en génomique comparative. Nos contributions ont consisté à déterminer les cas difficiles algorithmiquement, à étudier les différents moyens pour contourner cette difficulté, mais également à proposer des implémentations de certains de nos algorithmes.

En étudiant ces quelques problèmes à l'interface de la biologie et de l'algorithmique, j'ai appris à maîtriser différentes techniques pour l'obtention d'algorithmes de complexité paramétrée, plusieurs méthodes pour l'obtention de résultats de difficulté (l'élaboration de « gadgets »), mais aussi à ne pas rester bloquer sur une intuition se révélant fausse. Par exemple, j'ai longtemps cru qu'il serait possible d'obtenir un algorithme d'approximation à ratio constant pour le problème MAXIMUM GRAPH MOTIF lorsque le motif est colorful, le réseau est un arbre et chaque couleur n'apparaît que deux fois dans le réseau. En effet, les contraintes me semblaient suffisamment fortes pour qu'un tel algorithme soit envisageable. Cependant, la Proposition 5.2.3 a prouvé le contraire. Ainsi, comme mentionné dans [GJ79], il faut parfois chercher dans deux directions simultanément (un résultat positif ou un résultat négatif). Pour un problème donné, l'échec dans l'élaboration d'un algorithme permettant de le résoudre peut expliquer en partie sa difficulté algorithmique, et inversement, ne pas réussir à prouver la difficulté du problème peut néanmoins fournir des pistes pour construire un algorithme. Il ne faut donc pas hésiter à passer d'une ligne de recherche à l'autre.

Perspectives à court terme

Dans chaque chapitre de ce manuscrit, un certain nombre de questions ouvertes et de perspectives ont été proposées dans les « encarts ». Il est probable qu'une grande partie de mes recherches futures visera à essayer d'y répondre. En outre, il existe deux autres problèmes m'intéressant fortement et sur lesquels je travaillerai certainement à court terme.

– Une généralisation naturelle des intervalles sont les 2-intervalles, une union disjointe de deux intervalles sur les réels. En définissant un sommet par paire d'inter-

valles et une arête entre deux sommets si la paire d'intervalles est d'intersection non-vide, on obtient un graphe de 2-intervalles. Ces graphes peuvent notamment représenter une structure secondaire d'ARN (mais également de l'ordonnancement de tâches séparées en deux). Une étude préliminaire effectuée avec Romeo Rizzi et Stéphane Vialette montre que reconnaître si un graphe est un graphe de 2-intervalles unitaires (chaque intervalle est exactement de même taille) est un problème NP-Complet. Nous aimerions maintenant déterminer des bornes sur le plus petit entier d tel qu'un graphe d'intervalles puisse être un graphe de d-intervalles unitaires.

– Il est courant en génomique comparative de calculer la plus longue sous-séquence commune de deux séquences (représentant par exemple des génomes). Une variante consiste à chercher la plus longue sous-séquence commune avec la contrainte additionnelle qu'aucun caractère ne soit répété plus d'une fois. Une étude effectuée avec Guillaume Blin, Paola Bonizzoni et Riccardo Dondi montre qu'il n'existe pas de noyau ayant une taille polynomiale (sous de fortes hypothèses de complexité). De plus, nous avons montré qu'il est possible d'utiliser le problème MULTILINEAR DETECTION pour améliorer l'algorithme de complexité paramétrée existant. Ceci montre encore une fois la puissance potentielle de cette technique récente. Nous aimerions maintenant étudier l'existence d'un algorithme d'approximation à ratio constant, le problème ayant déjà été montré APX-Difficile.

Perspectives à plus long terme

Nous pouvons donner quelques idées générales récurrentes dans ce domaine de recherche qui lie les motivations biologiques à l'informatique théorique.

– Quand le problème est montré NP-Complet, la recherche d'algorithmes d'approximation, d'algorithmes de complexité paramétrée ou de règles de réduction doit être envisagé.
– S'il est prouvé qu'un problème n'appartient pas à la classe FPT pour un paramètre donné, chercher des paramètres moins naturels peut être envisagé. De manière similaire, les demandes en matière d'approximation peuvent être relâchées. Par exemple, s'il n'existe pas de schéma d'approximation, peut-être existe t-il un algorithme d'approximation à ratio constant ? S'il n'existe pas d'algorithme d'approximation à ratio constant, peut-être existe t-il un algorithme d'approximation dont le ratio dépend de la taille de l'entrée ?
– Si des algorithmes de complexité paramétrée ou d'approximation ont été montrés avec une complexité pire cas semblant trop importante pour pouvoir être implémenté, diminuer la complexité de ces algorithmes peut être envisagé, en utilisant de nouvelles techniques.
– Si la complexité semble raisonnable, implémenter les algorithmes correspondants

(en offrant une interface facile d'accès), les tester et les comparer peut être envisagé.

Bien sûr, ces pistes ne sont pas toujours toutes pertinentes selon le problème considéré. Néanmoins, étant donnée l'importance actuelle des problèmes relatifs aux réseaux biologiques et à la génomique comparative dans la biologie moderne, il semble important de contribuer au mieux à ce domaine, en utilisant les avancées de l'informatique théorique, collectées depuis de nombreuses années. Ces types de problèmes sont intéressants, motivants à étudier (encore plus à résoudre !) et utiles pour des communautés connexes de chercheurs – une alchimie très satisfaisante pour une thèse !

Liste des figures

Liste des tableaux

Index des problèmes

Bibliographie

[AA03] Eric ALM et Adam P. ARKIN : Biological Networks. *Current Opinion in Structural Biology*, 13(2) :193–202, 2003. Cité pages 69, 74, 129 et 130.

[ABB+00] Michael ASHBURNER, Catherine A. BALL, Judith A. BLAKE, David BOTSTEIN, Heather BUTLER, J. Michael CHERRY, Allan P. DAVIS, Kara DOLINSKI, Selina S. DWIGHT, Janan T. EPPIG, Midori A. HARRIS, David P. HILL, Laurie ISSEL-TARVER, Andrew KASARSKIS, Suzanna LEWIS, John C. MATESE, Joel E. RICHARDSON, Martin RINGWALD, Gerald M. RUBIN et Gavin SHERLOCK : Gene Ontology : tool for the unification of biology. *Nature genetics*, 25 :25–29, 2000. Cité page 91.

[ABRH+10] Abhimanyu M. AMBALATH, Radheshyam BALASUNDARAM, Chintan RAO H., Venkata KOPPULA, Neeldhara MISRA, Geevarghese PHILIP et M. S. RAMANUJAN : On the Kernelization Complexity of Colorful Motifs. *In* Venkatesh RAMAN et Saket SAURABH, éditeurs : *Proceedings of the 5th International Symposium Parameterized and Exact Computation (IPEC)*, volume 6478 de *Lecture Notes in Computer Science*, pages 14–25. Springer, 2010. Cité pages 92, 93 et 102.

[ACG+99] Giorgio AUSIELLO, Pierluigi CRESCENZI, Giorgio GAMBOSI, Viggo KANN, Alberto MARCHETTI-SPACCAMELA et Marco PROTASI : *Complexity and Approximation : Combinatorial Optimization Problems and Their Approximability Properties*. Springer, 1999. Cité page 57.

[ADH+08] Noga ALON, Phuong DAO, Iman HAJIRASOULIHA, Fereydoun HORMOZDIARI et Süleyman CENK SAHINALP : Biomolecular network motif counting and discovery by color coding. *In Proceedings of the 16th International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 241–249, 2008. Cité page 104.

[AFR+07] Sébastien ANGIBAUD, Guillaume FERTIN, Irena RUSU, Annelyse THÉVENIN et Stéphane VIALETTE : A Pseudo-boolean Programming Approach for Computing the Breakpoint Distance Between Two Genomes with Duplicate Genes. *In* Glenn TESLER et Dannie DURAND, éditeurs :

Proceedings of the RECOMB International Workshop Comparative Genomics (RECOMB-CG), volume 4751 de *Lecture Notes in Computer Science*, pages 16–29. Springer, 2007. Cité page 175.

[AFR08] Sébastien ANGIBAUD, Guillaume FERTIN et Irena RUSU : On the Approximability of Comparing Genomes with Duplicates. *In* Shin-Ichi NAKANO et Md. Saidur RAHMAN, éditeurs : *Proceedings of the Second International Workshop Algorithms and Computation (WALCOM)*, volume 4921 de *Lecture Notes in Computer Science*, pages 34–45. Springer, 2008. Cité pages 174, 175 et 176.

[AFR+09] Sébastien ANGIBAUD, Guillaume FERTIN, Irena RUSU, Annelyse THÉVENIN et Stéphane VIALETTE : On the Approximability of Comparing Genomes with Duplicates. *Journal of Graph Algorithms and Applications*, 13(1) :19–53, 2009. Cité page 175.

[AGGN04] Jochen ALBER, Jens GRAMM, Jiong GUO et Rolf NIEDERMEIER : Computing the similarity of two sequences with nested arc annotations. *Theoretical Computer Science*, 312(2-3) :337–358, 2004. Cité page 187.

[AJL+07] Bruce ALBERTS, Alexander JOHNSON, Julian LEWIS, Martin RAFF, Keith ROBERTS et Peter WALTER : *Molecular Biology of the Cell*. Garland Science, fifth édition, 2007. Cité pages 62 et 65.

[AYZ95] Noga ALON, Raphael YUSTER et Uri ZWICK : Color coding. *Journal of the ACM*, 42(4) :844–856, 1995. Cité pages 40 et 176.

[Bad03] Gary D. BADER : *Design and use of the Biomolecular Interaction Network Database (BIND) for Storing and Analyzing Protein-Protein Interaction Data*. Thèse de doctorat, University of Toronto, 2003. Cité page 72.

[BBD+12] Guillaume BLIN, Paola BONIZZONI, Riccardo DONDI, Romeo RIZZI et Florian SIKORA : Complexity Insights of the Minimum Duplication Problem. *In Proceedings of the 38th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'12)*, Lecture Notes in Computer Science. Springer, 2012. À paraître. Cité pages 15, 181 et 184.

[BBEW07] Mukul S. BANSAL, John Gordon BURLEIGH, Oliver EULENSTEIN et André WEHE : Heuristics for the Gene-Duplication Problem : A $\Theta(n)$ -Speed-Up for the Local Search. *In* Terence P. SPEED et Haiyan HUANG, éditeurs : *Proceedings of the 11th Annual International Conference Research in Computational Molecular Biology (RECOMB)*, volume 4453 de *Lecture Notes in Computer Science*, pages 238–252. Springer, 2007. Cité pages 182 et 184.

[BBL98] Prosenjit BOSE, Jonathan F. BUSS et Anna LUBIW : Pattern Matching for Permutations. *Information Processing Letters*, 65(5) :277–283, 1998. Cité page 189.

[BCF+07] Guillaume BLIN, Cedric CHAUVE, Guillaume FERTIN, Romeo RIZZI et Stéphane VIALETTE : Comparing Genomes with Duplications : A Computational Complexity Point of View. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 4(4) :523–534, 2007. Cité page 179.

[BCMR08] Anne BERGERON, Cedric CHAUVE, Fabien de MONTGOLFIER et Mathieu RAFFINOT : Computing Common Intervals of K Permutations, with Applications to Modular Decomposition of Graphs. *SIAM Journal on Discrete Mathematics*, 22(3) :1022–1039, 2008. Cité page 179.

[BCR02] Anne BERGERON, Sylvie CORTEEL et Mathieu RAFFINOT : The Algorithmic of Gene Teams. *In* Roderic GUIGÓ et Dan GUSFIELD, éditeurs : *Proceedings of the Second International Workshop Algorithms in Bioinformatics (WABI)*, volume 2452 de *Lecture Notes in Computer Science*, pages 464–476. Springer, 2002. Cité pages 178 et 179.

[BDFH09] Hans L. BODLAENDER, Rodney G. DOWNEY, Michael R. FELLOWS et Danny HERMELIN : On problems without polynomial kernels. *Journal of Computer and System Sciences*, 75(8) :423–434, 2009. Cité pages 53 et 102.

[BEW09] Mukul S. BANSAL, Oliver EULENSTEIN et André WEHE : The Gene-Duplication Problem : Near-Linear Time Algorithms for NNI-Based Local Searches. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 6(2) :221–231, 2009. Cité page 184.

[BFKN08] Nadja BETZLER, Michael R. FELLOWS, Christian KOMUSIEWICZ et Rolf NIEDERMEIER : Parameterized Algorithms and Hardness Results for Some Graph Motif Problems. *In* Paolo FERRAGINA et Gad M. LANDAU, éditeurs : *Proceedings of the 19th Annual Symposium Combinatorial Pattern Matching (CPM)*, volume 5029 de *Lecture Notes in Computer Science*, pages 31–43. Springer, 2008. Cité pages 92, 95, 98, 99, 114, 116, 135 et 198.

[BFMB+11] Guillaume BLIN, Guillaume FERTIN, Hafedh MOHAMED-BABOU, Irena RUSU, Florian SIKORA et Stéphane VIALETTE : Algorithmic Aspects of Heterogeneous Biological Networks Comparison. *In* Weifan WANG, Xuding ZHU et Ding-Zhu DU, éditeurs : *Proceedings of the 5th Annual International Conference on Combinatorial Optimization and Applications (COCOA'11)*, volume 6831 de *Lecture Notes in Computer Science*, pages 272–286. Springer, 2011. Cité pages 14, 155, 156, 157, 158, 162 et 201.

[BFSV09] Guillaume BLIN, Guillaume FERTIN, Florian SIKORA et Stéphane VIALETTE : The Exemplar Breakpoint Distance for Non-trivial Genomes Cannot Be Approximated. *In* Sandip DAS et Ryuhei UEHARA, éditeurs : *Proceedings of the Third International Workshop Algorithms and Computa-*

tion (WALCOM), volume 5431 de *Lecture Notes in Computer Science*, pages 357–368. Springer, 2009. Cité pages 15, 172, 175, 176 et 201.

[BGJ10] Jaroslaw BYRKA, Sylvain GUILLEMOT et Jesper JANSSON : New results on optimizing rooted triplets consistency. *Discrete Applied Mathematics*, 158(11) :1136–1147, 2010. Cité page 184.

[BHK+09] Sharon BRUCKNER, Falk HÜFFNER, Richard M. KARP, Ron SHAMIR et Roded SHARAN : Topology-Free Querying of Protein Interaction Networks. *In* Serafim BATZOGLOU, éditeur : *Proceedings of the 13th Annual International Conference Research in Computational Molecular Biology (RECOMB)*, volume 5541 de *Lecture Notes in Computer Science*, pages 74–89. Springer, 2009. Cité pages 91, 92, 95, 110, 111, 112, 137, 146, 147, 149, 150 et 151.

[BHKK07] Andreas BJÖRKLUND, Thore HUSFELDT, Petteri KASKI et Mikko KOIVISTO : Fourier meets möbius : fast subset convolution. *In* David S. JOHNSON et Uriel FEIGE, éditeurs : *Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC)*, pages 67–74. ACM, 2007. Cité pages 99, 100 et 104.

[Bix02] Robert E. BIXBY : Solving Real-World Linear Programs : A Decade and More of Progress. *Operations Research*, 50(1) :3–15, 2002. Cité page 51.

[BJMS09] Sebastian BÖCKER, Katharina JAHN, Julia MIXTACKI et Jens STOYE : Computation of median gene clusters. *Journal of Computational Biology*, 16(8) :1085–1099, 2009. Cité pages 178 et 179.

[BLW86] Norman L. BIGGS, E. Keith LLOYD et Robin J. WILSON : *Graph theory, 1736-1936*. Oxford University Press, USA, 1986. Cité page 23.

[BLW04] Johannes BERG, Michael LÄSSIG et Andreas WAGNER : Structure and evolution of protein interaction networks : a statistical model for link dynamics and gene duplications. *BMC Evolutionary Biology*, 4 :51, 2004. Cité page 75.

[BML+05] Frédéric BOYER, Anne MORGAT, Laurent LABARRE, Joël POTHIER et Alain VIARI : Syntons, metabolons and interactons : an exact graph-theoretical approach for exploring neighbourhood between genomic and functional data. *Bioinformatics*, 21(23) :4209–4215, 2005. Cité pages 82, 155 et 156.

[Bod09] Hans L. BODLAENDER : Kernelization : New Upper and Lower Bound Techniques. *In Proceedings of the Parameterized and Exact Computation, 4th International Workshop (IWPEC)*, volume 5917 de *Lecture Notes in Computer Science*, pages 17–37. Springer, 2009. Cité page 53.

[BRS09] Sebastian BÖCKER, Florian RASCHE et Tamara STEIJGER : Annotating Fragmentation Patterns. *In* Steven SALZBERG et Tandy WARNOW, éditeurs : *Proceedings of the 9th International Workshop Algorithms in Bioinfor-*

matics (WABI), volume 5724 de *Lecture Notes in Computer Science*, pages 13–24. Springer, 2009. Cité page 115.

[BRSV11a] Guillaume BLIN, Romeo RIZZI, Florian SIKORA et Stéphane VIALETTE : Minimum Mosaic Inference of a Set of Recombinants. *In* Alex POTANIN et Taso VIGLAS, éditeurs : *Proceedings of the 17th Computing : the Australasian Theory Symposium (CATS)*, volume 119 de *CRPIT*, pages 23–30. ACS, 2011. Cité pages 15, 168, 170, 171 et 172.

[BRSV11b] Guillaume BLIN, Romeo RIZZI, Florian SIKORA et Stéphane VIALETTE : Minimum Mosaic Inference of a Set of Recombinants. *International Journal of Foundations of Computer Science (IJFCS)*, 2011. Accepté pour publication. Cité page 168.

[Bry00] David BRYANT : The complexity of calculating exemplar distances. *In* David SANKOFF et Joseph H. NADEAU, éditeurs : *Comparative Genomics : Empirical and Analytical Approaches to Gene Order Dynamics, Map Alignment and Evolution of Gene Families*, pages 207–211. Kluwer, 2000. Cité page 174.

[BS06] Anne BERGERON et Jens STOYE : On the Similarity of Sets of Permutations and Its Applications to Genome Comparison. *Journal of Computational Biology*, 13(7) :1340–1354, 2006. Cité page 178.

[BS11] Mukul S. BANSAL et Ron SHAMIR : A Note on the Fixed Parameter Tractability of the Gene-Duplication Problem. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 8(3) :848–850, 2011. Cité pages 183 et 184.

[BSV09] Guillaume BLIN, Florian SIKORA et Stéphane VIALETTE : Querying Protein-Protein Interaction Networks. *In* Ion I. MANDOIU, Giri NARASIMHAN et Yanqing ZHANG, éditeurs : *Proceedings of the 5th International Symposium Bioinformatics Research and Applications (ISBRA)*, volume 5542 de *Lecture Notes in Bioinformatics*, pages 52–62. Springer-Verlag, 2009. Cité pages 14 et 84.

[BSV10a] Guillaume BLIN, Florian SIKORA et Stéphane VIALETTE : GraMoFoNe : a Cytoscape plugin for querying motifs without topology in Protein-Protein Interactions networks. *In* Hisham AL-MUBAID, éditeur : *Proceedings of the 2nd International Conference on Bioinformatics and Computational Biology (BICoB)*, pages 38–43. International Society for Computers and their Applications (ISCA), 2010. Cité pages 15, 138 et 149.

[BSV10b] Guillaume BLIN, Florian SIKORA et Stéphane VIALETTE : Querying Graphs in Protein-Protein Interactions Networks using Feedback Vertex Set. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 7(4) :628–635, 2010. Special Issue-ISBRA 2009-Bioinformatics Research and Applications. Cité pages 14, 84 et 86.

[CBB+10] Michael COSTANZO, Anastasia BARYSHNIKOVA, Jeremy BELLAY, Yungil KIM, Eric D. SPEAR, Carolyn S. SEVIER, Huiming DING, Judice L. Y. KOH, Kiana TOUFIGHI, Sara MOSTAFAVI, Jeany PRINZ, Robert P. ST. ONGE, Benjamin VANDERSLUIS, Taras MAKHNEVYCH, Franco J. VIZEACOUMAR, Solmaz ALIZADEH, Sondra BAHR, Renee L. BROST, Yiqun CHEN, Murat COKOL, Raamesh DESHPANDE, Zhijian LI, Zhen-Yuan LIN, Wendy LIANG, Michaela MARBACK, Jadine PAW, Bryan-Joseph SAN LUIS, Ermira SHUTERIQI, Amy H. TONG, Nydia van DYK, Iain M. WALLACE, Joseph A. WHITNEY, Matthew T. WEIRAUCH, Guoqing ZHONG, Hongwei ZHU, Walid A. HOURY, Michael BRUDNO, Sasan RAGIBIZADEH, Balázs PAPP, Csaba PÁL, Frederick P. ROTH, Guri GIAEVER, Corey NISLOW, Olga G. TROYANSKAYA, Howard BUSSEY, Gary D. BADER, Anne-Claude GINGRAS, Quaid D. MORRIS, Philip M. KIM, Chris A. KAISER, Chad L. MYERS, Brenda J. ANDREWS et Charles BOONE : The Genetic Landscape of a Cell. *Science*, 327(5964) :425–431, 2010. Cité page 129.

[CBFBE11] Wen-Chieh CHANG, John Gordon BURLEIGH, David F FERNÁNDEZ-BACA et Oliver EULENSTEIN : An ILP solution for the gene duplication problem. *BMC Bioinformatics (APBC 2011)*, 12(Suppl 1) :S14, 2011. Cité page 184.

[CC08] Liang-Hui CHU et Bor-Sen CHEN : Construction of a cancer-perturbed protein-protein interaction network for discovery of apoptosis drug targets. *BMC systems biology*, 2 :56+, 2008. Cité page 80.

[CFZ06] Zhixiang CHEN, Bin FU et Binhai ZHU : The Approximability of the Exemplar Breakpoint Distance Problem. *In* Siu-Wing CHENG et Chung Keung POON, éditeurs : *Proceedings of the Second International Conferenc Algorithmic Aspects in Information and Management (AAIM)*, volume 4041 de *Lecture Notes in Computer Science*, pages 291–302. Springer, 2006. Cité pages 175 et 176.

[CHM81] Michel CHEIN, Michel HABIB et Marie-Catherine MAURER : Partitive hypergraphs. *Discrete mathematics*, 37(1) :35–50, 1981. Cité pages 127 et 128.

[Cla05] David P. CLARK : *Molecular Biology*. Elvesier Academic Press, 2005. Cité pages 13, 61, 62, 68, 69 et 70.

[CLR+06] Carolyn R. CHO, Mark LABOW, Mischa REINHARDT, Jan van van OOSTRUM et Manuel C. PEITSCH : The application of systems biology to drug discovery. *Current opinion in chemical biology*, 10(4) :294–302, 2006. Cité page 80.

[CLRS02] Thomas H. CORMEN, Charles E. LEISERSON, Ronald L. RIVEST et Clifford STEIN : *Introduction à l'algorithmique : cours et exercices*. Dunod, seconde édition, 2002. Cité pages 29 et 158.

[CLSZ07] Jianer CHEN, Songjian LU, Sing-Hoi SZE et Fenghui ZHANG : Improved algorithms for path, matching, and packing problems. *In* Nikhil BANSAL, Kirk PRUHS et Clifford STEIN, éditeurs : *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 298–307. SIAM, 2007. Cité pages 42 et 43.

[Coo71] Stephen A. COOK : The Complexity of Theorem-Proving Procedures. *In Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 151–158, New York, 1971. ACM. Cité page 35.

[Cun82] William H. CUNNINGHAM : Decomposition of directed graphs. *SIAM Journal on Algebraic and Discrete Methods*, 3(2) :214–228, 1982. Cité page 135.

[CXR+05] Steve E. CALVANO, Wenzhong XIAO, Daniel R. RICHARDS, Ramon M. FELCIANO, Henry V. BAKER, Raymond J. CHO, Richard O. CHEN, Bernard H. BROWNSTEIN, J. Perren COBB, S. Kevin TSCHOEKE, Carol MILLER-GRAZIANO, Lyle L. MOLDAWER, Michael N. MINDRINOS, Ronald W. DAVIS, Ronald G. TOMPKINS, Stephen F. LOWRY et Large Scale COLLAB : A network-based analysis of systemic inflammation in humans. *Nature*, 437(7061) :1032–1037, 2005. Cité page 80.

[Dah00] Elias DAHLHAUS : Parallel algorithms for hierarchical clustering and applications to split decomposition and parity graph recognition. *Journal of Algorithms*, 36(2) :205–240, 2000. Cité page 136.

[Dar59] Charles DARWIN : *The Origin of Species by Means of Natural Selection*. John Murray, 1859. Cité page 74.

[DB06] Eugene DAVYDOV et Serafim BATZOGLOU : A computational model for RNA multiple structural alignment. *Theoretical Computer Science*, 368(3) : 205–216, 2006. Cité page 188.

[DF86] Martin E. DYER et Alan M. FRIEZE : Planar 3DM is NP-Complete. *Journal of Algorithms*, 7(2) :174–184, 1986. Cité page 191.

[DF99] Rodney G. DOWNEY et Michael R. FELLOWS : *Parameterized Complexity*. Springer-Verlag, 1999. Cité pages 37, 38, 50 et 52.

[DFV07] Riccardo DONDI, Guillaume FERTIN et Stéphane VIALETTE : Weak pattern matching in colored graphs : Minimizing the number of connected components. *In* Giuseppe F. ITALIANO, Eugenio MOGGI et Luigi LAURA, éditeurs : *Proceedings of the 10th Italian Conference Theoretical Computer Science (ICTCS)*, pages 27–38. World Scientific, 2007. Cité pages 115 et 116.

[DFV09] Riccardo DONDI, Guillaume FERTIN et Stéphane VIALETTE : Maximum Motif Problem in Vertex-Colored Graphs. *In* Gregory KUCHEROV et Esko

UKKONEN, éditeurs : *Proceedings of the 20th Annual Symposium Combinatorial Pattern Matching (CPM)*, volume 5577 de *Lecture Notes in Computer Science*, pages 221–235. Springer, 2009. Cité pages 111 et 118.

[DFV11] Riccardo DONDI, Guillaume FERTIN et Stéphane VIALETTE : Finding Approximate and Constrained Motifs in Graphs. *In* Raffaele GIANCARLO et Giovanni MANZINI, éditeurs : *Proceedings of the 22nd Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 6661 de *Lecture Notes in Computer Science*, pages 388–401. Springer, 2011. Cité page 122.

[DH09] Erik D. DEMAINE et Robert A. HEARN : Playing Games with Algorithms : Algorithmic Combinatorial Game Theory. *In* Michael H. ALBERT et Richard J. NOWAKOWSKI, éditeurs : *Games of No Chance 3*, volume 56 de *Mathematical Sciences Research Institute Publications*, pages 3–56. Cambridge University Press, 2009. Cité page 35.

[Die05] Reinhard DIESTEL : *Graph Theory*. Numéro 173 de Graduate texts in Mathematics. Springer-Verlag, third édition, 2005. Cité pages 19 et 84.

[DMER09] Geraldine DEL MONDO, Damien EVEILLARD et Irena RUSU : Homogeneous decomposition of protein interaction networks : refining the description of intra-modular interactions. *Bioinformatics*, 25(7) :926–932, 2009. Cité page 135.

[DS04] Irit DINUR et Samuel SAFRA : On the Hardness of Approximating Minimum Vertex Cover. *Annals of Mathematics*, 162 :2005, 2004. Cité page 57.

[DSG+07] Banu DOST, Tomer SHLOMI, Nitin GUPTA, Eytan RUPPIN, Vineet BAFNA et Roded SHARAN : QNet : A Tool for Querying Protein Interaction Networks. *In* Terence P. SPEED et Haiyan HUANG, éditeurs : *Proceedings of the 11th Annual International Conference Research in Computational Molecular Biology (RECOMB)*, volume 4453 de *Lecture Notes in Computer Science*, pages 1–15. Springer, 2007. Cité pages 83 et 86.

[DT97] George B. DANTZIG et Mukund N. THAPA : *Linear programming 1 : introduction*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997. Cité pages 58 et 59.

[DVLMS06] Barbara DI VENTURA, Caroline LEMERLE, Konstantinos MICHALODIMITRAKIS et Luis SERRANO : From in vivo to in silico biology and back. *Nature*, 443(7111) :527–533, 2006. Cité page 80.

[Edd04] Sean R. EDDY : What is dynamic programming ? *Nature Biotechnology*, 22(7) :909–910, July 2004. Cité page 39.

[EKJ+02] Aled M. EDWARDS, Bart KUS, Ronald JANSEN, Dov GREENBAUM, Jack GREENBLATT et Mark GERSTEIN : Bridging structural biology and geno-

mics : assessing protein interaction data with known complexes. *Trends in Genetics*, 18(10) :529 – 536, 2002. Cité page 72.

[ES03] Evan E. EICHLER et David SANKOFF : Structural Dynamics of Eukaryotic Chromosome Evolution. *Science*, 301(5634) :793–797, 2003. Cité page 181.

[Eul36] Leonhard EULER : Solutio problematis ad geometriam situs pertinentis. *Commentarii Academiae Scientiarum Imperialis Petropolitanae*, 8 :128–140, 1736. Cité page 23.

[Eva99] Patricia A. EVANS : Finding Common Subsequences with Arcs and Pseudoknots. *In* Maxime CROCHEMORE et Mike PATERSON, éditeurs : *Proceedings of the 10th Annual Symposium Combinatorial Pattern Matching (CPM)*, volume 1645 de *Lecture Notes in Computer Science*, pages 270–280. Springer, 1999. Cité pages 187 et 188.

[Eva06] Patricia A. EVANS : Finding Common RNA Pseudoknot Structures in Polynomial Time. *In* Moshe LEWENSTEIN et Gabriel VALIENTE, éditeurs : *Proceedings of the 17th Annual Symposium Combinatorial Pattern Matching (CPM)*, volume 4009 de *Lecture Notes in Computer Science*, pages 223–232. Springer, 2006. Cité pages 188, 189 et 190.

[FAB+11] Paul FLICEK, M. Ridwan AMODE, Daniel BARRELL, Kathryn BEAL, Simon BRENT, Yuan CHEN, Peter CLAPHAM, Guy COATES, Susan FAIRLEY, Stephen FITZGERALD, Leo GORDON, Maurice HENDRIX, Thibaut HOURLIER, Nathan JOHNSON, Andreas KÄHÄRI, Damian KEEFE, Stephen KEENAN, Rhoda KINSELLA, Felix KOKOCINSKI, Eugene KULESHA, Pontus LARSSON, Ian LONGDEN, William MCLAREN, Bert OVERDUIN, Bethan PRITCHARD, Harpreet Singh S. RIAT, Daniel RIOS, Graham R. RITCHIE, Magali RUFFIER, Michael SCHUSTER, Daniel SOBRAL, Giulietta SPUDICH, Y. Amy TANG, Stephen TREVANION, Jana VANDROVCOVA, Albert J. VILELLA, Simon WHITE, Steven P. WILDER, Amonida ZADISSA, Jorge ZAMORA, Bronwen L. AKEN, Ewan BIRNEY, Fiona CUNNINGHAM, Ian DUNHAM, Richard DURBIN, Xosé M. FERNÁNDEZ-SUAREZ, Javier HERRERO, Tim J. HUBBARD, Anne PARKER, Glenn PROCTOR, Jan VOGEL et Stephen M. SEARLE : Ensembl 2011. *Nucleic Acids Research*, 39(Database issue) :D800–D806, 2011. Cité page 149.

[Fel01] Michael R. FELLOWS : Parameterized Complexity : The Main Ideas and Some Research Frontiers. *In* Peter EADES et Tadao TAKAOKA, éditeurs : *Proceedings of the 12th International Symposium on Algorithms and Computation (ISAAC)*, volume 2223 de *Lecture Notes in Computer Science*, pages 291–307. Springer, 2001. Cité page 39.

[Fel02] Michael R. FELLOWS : Parameterized Complexity : The Main Ideas and Connections to Practical Computing. *In* Rudolf FLEISCHER, Bernard M. E.

MORET et Erik Meineche SCHMIDT, éditeurs : *Experimental Algorithmics, From Algorithm Design to Robust and Efficient Software [Dagstuhl seminar, September 2000]*, volume 2547 de *Lecture Notes in Computer Science*, pages 51–77. Springer, 2002. Cité page 56.

[Fes09] Paola FESTA : Matroids. *In* Christodoulos A. FLOUDAS et Panos M. PARDALOS, éditeurs : *Encyclopedia of Optimization*, pages 1975–1981. Springer, 2009. Cité page 98.

[FFHV07] Michael R. FELLOWS, Guillaume FERTIN, Danny HERMELIN et Stéphane VIALETTE : Sharp tractability borderlines for finding connected motifs in vertex-colored graphs. *In* Lars ARGE, Christian CACHIN, Tomasz JURDZINSKI et Andrzej TARLECKI, éditeurs : *Proceedings of the 34th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 4596 de *Lecture Notes in Computer Science*, pages 340–351, Poland, 2007. Springer. Cité pages 90, 92, 93, 94, 95, 98 et 133.

[FG04] Jörg FLUM et Martin GROHE : The parameterized complexity of counting problems. *SIAM Journal on Computing*, 33(4) :892–922, 2004. Cité page 104.

[FG06] Jörg FLUM et Martin GROHE : *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer Verlag, 2006. Cité pages 37 et 104.

[FK10] Fedor V. FOMIN et Dieter KRATSCH : *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2010. Cité page 60.

[FMBR11] Guillaume FERTIN, Hafedh MOHAMED BABOU et Irena RUSU : A Pattern-guided Approach to compare Heterogeneous Networks. `http://pagesperso.lina.univ-nantes.fr/~E09D478T/SGM-DB.pdf`. Soumis, 2011. Cité page 156.

[Gas10] Serge GASPERS : *Exponential Time Algorithms - Structures, Measures, and Bounds*. VDM, 2010. Cité page 60.

[GCM+79] Morris GOODMAN, John CZELUSNIAK, G. William MOORE, A. E. ROMERO-HERRERA et Genji MATSUDA : Fitting the Gene Lineage into its Species Lineage, a Parsimony Strategy Illustrated by Cladograms Constructed from Globin Sequences. *Systematic Zoology*, 28(2) :132–163, 1979. Cité page 182.

[GGH+06] Jiong GUO, Jens GRAMM, Falk HÜFFNER, Rolf NIEDERMEIER et Sebastian WERNICKE : Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization. *Journal of Computer and System Sciences*, 72(8) :1386–1396, 2006. Cité page 85.

[GIP99] Deborah GOLDMAN, Sorin ISTRAIL et Christos H. PAPADIMITRIOU : Algorithmic aspects of protein structure similarity. *In Proceedings of the*

40th Annual Symposium of Foundations of Computer Science (FOCS), pages 512–522. IEEE Computer Society, 1999. Cité page 189.

[GJ79] Michael R. GAREY et David S. JOHNSON : *Computers and Intractability : A guide to the theory of NP-completeness*. W.H. Freeman, San Francisco, 1979. Cité pages 31, 32, 33, 35, 37, 48, 85, 93, 130, 159, 190, 193 et 197.

[GKBC04] Julien GAGNEUR, Roland KRAUSE, Tewis BOUWMEESTER et Georg CASARI : Modular decomposition of protein-protein interaction networks. *Genome Biology*, 5(8) :R57, 2004. Cité pages 129 et 135.

[GKSGBJ11] Anthony GITTER, Judith KLEIN-SEETHARAMAN, Anupam GUPTA et Ziv BAR-JOSEPH : Discovering pathways by orienting edges in protein interaction networks. *Nucleic Acids Research*, 39(4) :e22, 2011. Cité pages 155 et 156.

[Gra76] Mark GRANOVETTER : Network Sampling : Some First Steps. *The American Journal of Sociology*, 81(6) :1287–1303, 1976. Cité page 92.

[GS09] Mira GONEN et Yuval SHAVITT : Approximating the Number of Network Motifs. *In* Konstantin AVRACHENKOV, Debora DONATO et Nelly LITVAK, éditeurs : *Proceedings of the 6th International Workshop Algorithms and Models for the Web-Graph (WAW)*, volume 5427 de *Lecture Notes in Computer Science*, pages 13–24. Springer, 2009. Cité page 104.

[GS10] Sylvain GUILLEMOT et Florian SIKORA : Finding and counting vertex-colored subtrees. *In* Petr HLINENÝ et Antonín KUCERA, éditeurs : *Proceedings of the 35th International Symposium on Mathematical Foundations of Computer Science (MFCS'10)*, volume 6281 de *Lecture Notes in Computer Science*, pages 405–416, Brno, Czech Republic, August 2010. Springer. Cité pages 14, 96, 100, 103, 106, 107, 112 et 117.

[GS11] Sylvain GUILLEMOT et Florian SIKORA : Finding and counting vertex-colored subtrees. *Algorithmica*, 2011. Accepté pour publication. Cité pages 103, 105, 106, 114 et 150.

[GSS10] Iftah GAMZU, Danny SEGEV et Roded SHARAN : Improved Orientations of Physical Networks. *In* Vincent MOULTON et Mona SINGH, éditeurs : *Proceedings of the 10th International Workshop on Algorithms in Bioinformatics (WABI)*, volume 6293 de *Lecture Notes in Bioinformatics*, pages 215–225. Springer, 2010. Cité page 155.

[Gus97] Dan GUSFIELD : *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*. Cambridge University Press, 1997. Cité pages 40 et 79.

[HD05] Rose HOBERMAN et Dannie DURAND : The Incompatible Desiderata of Gene Cluster Properties. *In* Aoife MCLYSAGHT et Daniel H. HUSON,

éditeurs : *Proceedings of the 3rd RECOMB 2005 International Workshop Comparative Genomics (RCG)*, volume 3678 de *Lecture Notes in Computer Science*, pages 73–87. Springer, 2005. Cité pages 177 et 179.

[Her09] Danny HERMELIN : *New Results in Parameterized Complexity*. Thèse de doctorat, University of Haifa, 2009. Cité pages 36 et 197.

[HMP04] Michel HABIB, Fabien de MONTGOLFIER et Christophe PAUL : A Simple Linear-Time Modular Decomposition Algorithm for Graphs, Using Order Extension. *In* Torben HAGERUP et Jyrki KATAJAINEN, éditeurs : *Proceedings of the 9th Scandinavian Workshop on Algorithm Theory (SWAT)*, volume 3111 de *Lecture Notes in Computer Science*, pages 187–198. Springer, 2004. Cité page 128.

[HWZ08] Falk HÜFFNER, Sebastian WERNICKE et Thomas ZICHNER : Algorithm Engineering for Color-Coding with Applications to Signaling Pathway Detection. *Algorithmica*, 52(2) :114–132, 2008. Cité pages 42 et 99.

[IS08] Trey IDEKER et Roded SHARAN : Protein networks in disease. *Genome research*, 18(4) :644–652, 2008. Cité page 80.

[Jef99] Constance J. JEFFERY : Moonlighting proteins. *Trends in biochemical sciences*, 24(1) :8–11, 1999. Cité page 67.

[Jia10] Minghui JIANG : The Zero Exemplar Distance Problem. *In* Eric TANNIER, éditeur : *Proceedings of the RECOMB International Workshop Comparative Genomics (RECOMB-CG)*, volume 6398 de *Lecture Notes in Computer Science*, pages 74–82. Springer, 2010. Cité page 176.

[JLMZ04] Tao JIANG, Guohui LIN, Bin MA et Kaizhong ZHANG : The longest common subsequence problem for arc-annotated sequences. *Journal of Discrete Algorithms*, 2(2) :257–270, 2004. Cité page 187.

[JMBO01] Hawoong JEONG, Sean P. MASON, Albert-Laszlo BARABÁSI et Zoltan N. OLTVAI : Lethality and centrality in protein networks. *Nature*, 411(6833) : 41–42, 2001. Cité page 74.

[JS08] Björn H. JUNKER et Falk SCHREIBER : *Analysis of Biological Networks*. Wiley-Interscience, New York, NY, USA, 2008. Cité pages 69 et 70.

[JTA+00] Hawoong JEONG, B. TOMBOR, Reka ALBERT, Zoltan N. OLTVAI et Albert-Laszlo BARABÁSI : The large-scale organization of metabolic networks. *Nature*, 407(6804) :651–654, 2000. Cité page 74.

[Kar72] Richard M. KARP : Reducibility among combinatorial problems. *In* J.W. THATCHER et R.E MILLER, éditeurs : *Complexity of computer computations*, pages 85–103. Plenum Press, New York, 1972. Cité pages 35 et 40.

[Kar82] Richard M. KARP : Dynamic programming meets the principle of inclusion and exclusion. *Operations Research Letters*, 1(2) :49 – 51, 1982. Cité page 105.

[Kar09] George KARAKOSTAS : A better approximation ratio for the vertex cover problem. *ACM Transactions on Algorithms*, 5(4) :41 :1–41 :8, 2009. Cité page 57.

[KBS08] Maxim KALAEV, Vineet BAFNA et Roded SHARAN : Fast and Accurate Alignment of Multiple Protein Networks. *In Proceedings of the 12th Annual International Conference Research in Computational Molecular Biology (RECOMB)*, pages 246–256, 2008. Cité page 82.

[KG98] John D. KECECIOGLU et Dan GUSFIELD : Reconstructing a History of Recombinations From a Set of Sequences. *Discrete Applied Mathematics*, 88(1-3) :239–260, 1998. Cité pages 168 et 169.

[KGS05] Mehmet KOYUTÜRK, Ananth GRAMA et Wojciech SZPANKOWSKI : Pairwise Local Alignment of Protein Interaction Networks Guided by Models of Evolution. *In* Satoru MIYANO, Jill P. MESIROV, Simon KASIF, Sorin ISTRAIL, Pavel A. PEVZNER et Michael S. WATERMAN, éditeurs : *Proceedings of the 9th Annual International Conference Research in Computational Molecular Biology (RECOMB)*, Lecture Notes in Computer Science, pages 48–65, 2005. Cité page 81.

[Kit02] Hiroaki KITANO : Systems Biology : A Brief Overview. *Science*, 295(5560) : 1662–1664, 2002. Cité page 69.

[KMRR06] Joachim KNEIS, Daniel MÖLLE, Stefan RICHTER et Peter ROSSMANITH : Divide-and-Color. *In* Fedor V. FOMIN, éditeur : *Proceedings of the 32nd International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, volume 4271 de *Lecture Notes in Computer Science*, pages 58–67. Springer, 2006. Cité pages 42 et 43.

[Knu76] Donald E. KNUTH : Big Omicron and big Omega and big Theta. *SIGACT News*, 8(2) :18–24, April 1976. Cité page 29.

[Kou08] Ioannis KOUTIS : Faster Algebraic Algorithms for Path and Packing Problems. *In* Luca ACETO, Ivan DAMGÅRD, Leslie Ann GOLDBERG, Magnús M. HALLDÓRSSON, Anna INGÓLFSDÓTTIR et Igor WALUKIEWICZ, éditeurs : *Proceedings of the 35th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 5125 de *Lecture Notes in Computer Science*, pages 575–586. Springer, 2008. Cité page 43.

[KRU04] Mikko KOIVISTO, Pasi RASTAS et Esko UKKONEN : Recombination Systems. *In* Juhani KARHUMÄKI, Hermann A. MAURER, Gheorghe PAUN et Grzegorz ROZENBERG, éditeurs : *Theory Is Forever, Essays Dedicated to Arto Salomaa on the Occasion of His 70th Birthday*, volume 3113 de *Lecture Notes in Computer Science*, pages 159–169. Springer, 2004. Cité pages 168 et 169.

[KRVW06] Marcin KUBICA, Romeo RIZZI, Stéphane VIALETTE et Tomasz WALEN : Approximation of RNA Multiple Structural Alignment. *In* Moshe LE-

WENSTEIN et Gabriel VALIENTE, éditeurs : *Proceedings of the 17th Annual Symposium Combinatorial Pattern Matching (CPM)*, volume 4009 de *Lecture Notes in Computer Science*. Springer, 2006. Cité pages 188 et 189.

[KSK+03] Brian P. KELLEY, Roded SHARAN, Richard M. KARP, Taylor SITTLER, David E. ROOT, Brent R. STOCKWELL et Trey IDEKER : Conserved pathways within bacteria and yeast as revealed by global protein network alignment. *Proceedings of the National Academy of Sciences of the USA*, 100(20) :11394–11399, 2003. Cité pages 81 et 83.

[KT05] Jon KLEINBERG et Éva TARDOS : *Algorithm Design*. Addison-Wesley, 2005. Cité page 185.

[KW09] Ioannis KOUTIS et Ryan WILLIAMS : Limits and Applications of Group Algebras for Parameterized Problems. *In* Susanne ALBERS, Alberto MARCHETTI-SPACCAMELA, Yossi MATIAS, Sotiris E. NIKOLETSEAS et Wolfgang THOMAS, éditeurs : *Proceedings of the 36th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 5555 de *Lecture Notes in Computer Science*, pages 653–664. Springer, 2009. Cité pages 43 et 44.

[Lac07] Vincent LACROIX : *Identification de motifs dans les réseaux métaboliques*. Thèse de doctorat, Université Claude Bernard - Lyon I, 2007. Cité pages 74, 92 et 137.

[LBP07] Daniel LE BERRE et Anne PARRAIN : On extending SAT solvers for PB problems. *In Proceedings of the 14th RCRA workshop Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion (RCRA)*, Rome, 2007. Cité page 138.

[LDAM04] Insuk LEE, Shailesh V. DATE, Alex T. ADAI et Edward M. MARCOTTE : A probabilistic functional network of yeast genes. *Science*, 306 :1555–1558, 2004. Cité pages 82 et 155.

[Lev73] Leonid A. LEVIN : Universal sorting problems. *Problemy Predaci Informacii*, 9(3) :265–266, 1973. (In Russian). Cité page 35.

[LFS06] Vincent LACROIX, Cristina G. FERNANDES et Marie-France SAGOT : Motif search in graphs : application to metabolic networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 3(4) :360–368, 2006. Cité pages 13, 90, 91, 94, 98, 111, 114 et 137.

[LLB+09] Chung-Shou LIAO, Kanghao LU, Michael BAYM, Rohit SINGH et Bonnie BERGER : IsoRankN : spectral methods for global alignment of multiple protein networks. *Bioinformatics*, 25(12) :i253–i258, 2009. Cité page 81.

[LOO11] Mark S. LONGO, Michael J. O'NEILL et Rachel J. O'NEILL : Abundant Human DNA Contamination Identified in Non-Primate Genome Databases. *PLoS ONE*, 6(2) :e16410, Feb 2011. Cité page 64.

[LV04] Antoni LOZANO et Gabriel VALIENTE : On the Maximum Common Embedded Subtree Problem for Ordered Trees. *In* Costas S. ILIOPOULOS et Thierry LECROQ, éditeurs : *String Algorithmics*, chapitre 7, pages 155–169. King's College London Publications, 2004. Cité pages 188 et 189.

[MBZS08] Alexander MEDVEDOVSKY, Vineet BAFNA, Uri ZWICK et Roded SHARAN : An Algorithm for Orienting Graphs Based on Cause-Effect Pairs and Its Applications to Orienting Protein Networks. *In* Keith A. CRANDALL et Jens LAGERGREN, éditeurs : *Proceedings of the 8th International Workshop on Algorithms in Bioinformatics (WABI)*, volume 5251 de *Lecture Notes in Bioinformatics*, pages 222–232. Springer, 2008. Cité pages 155 et 156.

[MLZ00] Bin MA, Ming LI et Louxin ZHANG : From Gene Trees to Species Trees. *SIAM Journal on Computing*, 30(3) :729–752, 2000. Cité page 183.

[Mon03] Fabien de MONTGOLFIER : *Décomposition modulaire des graphes. Théorie, extensions et algorithmes*. Thèse de doctorat, Université Montpellier II, 2003. Cité page 127.

[Mor03] David MORRISON : Carl Woese and New Perspectives on Evolution. NASA Astrobiology Institute, 2003. Cité page 75.

[MR84] Rolf H. MÖHRING et Franz J. RADERMACHER : Substitution decomposition for discrete structures and connections with combinatorial optimization. *Annals of Discrete Mathematics*, 19 :257–355, 1984. Cité page 127.

[MVG+09] Hamish MCWILLIAM, Franck VALENTIN, Mickael GOUJON, Weizhong LI, Menaka NARAYANASAMY, Jenny MARTIN, Teresa MIYAR et Rodrigo LOPEZ : Web services at the European Bioinformatics Institute-2009. *Nucleic acids research*, 37(Web Server issue) :W6–10, 2009. Cité page 147.

[MWB09] Mathias MÖHL, Sebastian WILL et Rolf BACKOFEN : Lifting Prediction to Alignment of RNA Pseudoknots. *In* Serafim BATZOGLOU, éditeur : *Proceedings of the 13th Annual International Conference Research in Computational Molecular Biology (RECOMB)*, volume 5541 de *Lecture Notes in Computer Science*, pages 285–301. Springer, 2009. Cité page 188.

[MWK08] Jochen MALINOWSKI, Tim WEITZEL et Tobias KEIM : Decision support for team staffing : An automated relational recommendation approach. *Decision Support Systems*, 45(3) :429–447, 2008. Cité page 92.

[Ned09] Jesper NEDERLOF : Fast Polynomial-Space Algorithms Using Möbius Inversion : Improving on Steiner Tree and Related Problems. *In* Susanne ALBERS, Alberto MARCHETTI-SPACCAMELA, Yossi MATIAS, Sotiris E. NIKOLETSEAS et Wolfgang THOMAS, éditeurs : *Proceedings of the 36th International Colloquium Automata, Languages and Programming (ICALP)*,

volume 5555 de *Lecture Notes in Computer Science*, pages 713–725. Springer, 2009. Cité page 105.

[Nie06] Rolf NIEDERMEIER : *Invitation to Fixed Parameter Algorithms*. Lecture Series in Mathematics and Its Applications. Oxford University Press, 2006. Cité pages 37, 45, 46, 51, 52, 53, 84 et 126.

[NT84] Joseph H. NADEAU et Benjamin A. TAYLOR : Lengths of chromosomal segments conserved since divergence of man and mouse. *Proceedings of the National Academy of Sciences of the United States of America*, 81(3) :814–818, 1984. Cité page 178.

[OSC10] Aïda OUANGRAOUA, Krister M. SWENSON et Cedric CHAUVE : An Approximation Algorithm for Computing a Parsimonious First Speciation in the Gene Duplication Model. *In* Eric TANNIER, éditeur : *Proceedings of the RECOMB International Workshop Comparative Genomics (RECOMB-CG)*, volume 6398 de *Lecture Notes in Computer Science*, pages 290–301. Springer, 2010. Cité pages 182 et 184.

[Pau06] Christophe PAUL : *Aspects algorithmiques de la décomposition modulaire*. Habilitation à diriger des recherches, Université de Montpellier 2, 2006. Cité page 136.

[PBR+05] Sophie PASEK, Anne BERGERON, Jean-Loup RISLER, Alexandra LOUIS, Emmanuelle OLLIVIER et Mathieu RAFFINOT : Identification of genomic features using microsyntenies of domains : Domain teams. *Genome Research*, 15(6) :867–874, 2005. Cité page 179.

[Pev06] Pavel A. PEVZNER : *Bio-informatique Moléculaire. Une approche algorithmique.* Springer, 2006. Cité pages 66 et 79.

[PLM09] Annick PIERCE, Dominique LEGRAND et Joël MAZURIER : La lactoferrine : une protéine multifonctionnellen. *Médecine sciences*, 25(4) :361–369, 2009. Cité page 67.

[PRYLZU05] Ron Y. PINTER, Oleg ROKHLENKO, Esti YEGER-LOTEM et Michal ZIV-UKELSON : Alignment of metabolic pathways. *Bioinformatics*, 21(16) :3401–3408, 2005. Cité pages 83 et 156.

[PY91] Christos H. PAPADIMITRIOU et Mihalis YANNAKAKIS : Optimization, approximation and complexity classes. *Journal of Computer and System Sciences*, 43(3) :425–440, 1991. Cité pages 56, 160 et 162.

[RB09] Andrea ROLI et Christian BLUM : Tabu Search for the Founder Sequence Reconstruction Problem : A Preliminary Study. *In* Sigeru OMATU, Miguel ROCHA, José BRAVO, Florentino Fernández RIVEROLA, Emilio CORCHADO, Andrés BUSTILLO et Juan M. CORCHADO, éditeurs : *Proceedings of the 10th International Work-Conference on Artificial Neural Networks*

(IWANN), volume 5518 de *Lecture Notes in Computer Science*, pages 1035–1042. Springer, 2009. Cité page 171.

[RK06] Sven RAHMANN et Gunnar W. KLAU : Integer Linear Programs for Discovering Approximate Gene Clusters. *In* Philipp BUCHER et Bernard M. E. MORET, éditeurs : *Proceedings of the 6th International Workshop Algorithms in Bioinformatics (WABI)*, volume 4175 de *Lecture Notes in Computer Science*, pages 298–309. Springer, 2006. Cité pages 178 et 179.

[RS97] Ran RAZ et Shmuel SAFRA : A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. *In Proceedings of the 29th annual ACM Symposium on Theory of Computing (STOC)*, pages 475–484. ACM, 1997. Cité page 126.

[RSM+02] Erzsebet RAVASZ, A. L. SOMERA, D. A. MONGRU, Zoltan N. OLTVAI et Albert-Laszlo BARABASI : Hierarchical Organization of Modularity in Metabolic Networks. *Science*, 297(5586) :1551–1555, 2002. Cité page 129.

[RSV04] Bruce A. REED, Kaleigh SMITH et Adrian VETTA : Finding odd cycle transversals. *Operations Research Letters*, 32(4) :299–301, 2004. Cité page 46.

[RU07] Pasi RASTAS et Esko UKKONEN : Haplotype Inference Via Hierarchical Genotype Parsing. *In* Raffaele GIANCARLO et Sridhar HANNENHALLI, éditeurs : *Proceedings of the 7th International Workshop Algorithms in Bioinformatics (WABI)*, volume 4645 de *Lecture Notes in Computer Science*, pages 85–97. Springer, 2007. Cité pages 168, 169, 170 et 171.

[San99] David SANKOFF : Genome rearrangement with gene families. *Bioinformatics*, 15(11) :909–917, 1999. Cité page 173.

[SB05] David W. STAPLE et Samuel E. BUTCHER : Pseudoknots : RNA structures with diverse functions. *PLoS biology*, 3(6) :956–959, 2005. Cité pages 65 et 189.

[SHB+09] Damian SMEDLEY, Syed HAIDER, Benoit BALLESTER, Richard HOLLAND, Darin LONDON, Gudmundur THORISSON et Arek KASPRZYK : BioMart - biological queries made easy. *BMC genomics*, 10 :22, 2009. Cité page 149.

[SI06] Roded SHARAN et Trey IDEKER : Modeling cellular machinery through biological network comparison. *Nature biotechnology*, 24(4) :427–433, April 2006. Cité pages 72, 79, 80, 81, 82 et 201.

[SIKS06] Jacob SCOTT, Trey IDEKER, Richard M. KARP et Roded SHARAN : Efficient Algorithms for Detecting Signaling Pathways in Protein Interaction Networks. *Journal of Computational Biology*, 13(2) :133–144, 2006. Cité page 40.

[Ski08] Steven S. SKIENA : *The Algorithm Design Manual*. Springer Publishing Company, Incorporated, second édition, 2008. Cité pages 29 et 30.

[SLS09] Sophie SCHBATH, Vincent LACROIX et Marie-France SAGOT : Assessing the exceptionality of coloured motifs in networks. *EURASIP Journal on Bioinformatics and Systems Biology*, 2009 :1–9, 2009. Cité page 104.

[SMO+03] Paul SHANNON, Andrew MARKIEL, Owen OZIER, Nitin S. BALIGA, Jonathan T. WANG, Daniel RAMAGE, Nada AMIN, Benno SCHWIKOWSKI et Trey IDEKER : Cytoscape : A Software Environment for Integrated Models of Biomolecular Interaction Networks. *Genome Research*, 13 :2498–2504, 2003. Cité page 138.

[SSBD03] Steffen SCHMIDT, Shamil SUNYAEV, Peer BORK et Thomas DANDEKAR : Metabolites : a helping hand for pathway evolution ? *TRENDS in Biochemical Sciences*, 28(6) :336–341, 2003. Cité page 75.

[SSK+05] Roded SHARAN, Silpa SUTHRAM, Ryan M. KELLEY, Tanja KUHN, Scott MCCUINE, Peter UETZ, Taylor SITTLER, Richard M. KARP et Trey IDEKER : Conserved patterns of protein interaction in multiple species. *Proceedings of the National Academy of Sciences of the USA*, 102(6) :1974–1979, 2005. Cité pages 81 et 156.

[SSR+03] Eran SEGAL, Michael SHAPIRA, Aviv REGEV, Dana PE'ER, David BOTSTEIN, Daphne KOLLER et Nir FRIEDMAN : Module networks : identifying regulatory modules and their condition-specific regulators from gene expression data. *Nature genetics*, 34(2) :166–176, 2003. Cité page 129.

[SSRS06] Tomer SHLOMI, Daniel SEGAL, Eytan RUPPIN et Roded SHARAN : QPath : a method for querying pathways in a protein-protein interaction network. *BMC Bioinformatics*, 7 :199, 2006. Cité page 83.

[Ste99] Ulrike STEGE : Gene Trees and Species Trees : The Gene-Duplication Problem is Fixed-Parameter Tractable. *In* Frank K. H. A. DEHNE, Arvind GUPTA, Jörg-Rüdiger SACK et Roberto TAMASSIA, éditeurs : *Proceedings of the 6th International Workshop on Algorithms and Data Structures (WADS)*, volume 1663 de *Lecture Notes in Computer Science*, pages 288–293. Springer, 1999. Cité page 184.

[The11] THE UNIPROT CONSORTIUM : Ongoing and future developments at the Universal Protein Resource. *Nucleic Acids Research*, 39(Database Issue) : D214–D219, 2011. Cité pages 147 et 149.

[Tho09] Stéphan THOMASSÉ : A quadratic kernel for feedback vertex set. *In* Claire MATHIEU, éditeur : *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 115–119. SIAM, 2009. Cité page 85.

[TNTZ05] Cam THACH NGUYEN, Y. C. TAY et Louxin ZHANG : Divide-and-conquer approach for the exemplar breakpoint distance. *Bioinformatics*, 21(10) : 2171–2176, 2005. Cité page 175.

[UDZ+06] Peter UETZ, Yu-An DONG, Christine ZERETZKE, Christine ATZLER, Armin BAIKER, Bonnie BERGER, Seesandra V. RAJAGOPALA, Maria ROUPELIEVA, Dietlind ROSE, Even FOSSUM et Jürgen HAAS : Herpesviral Protein Networks and Their Interaction with the Human Proteome. *Science*, 311(5758) :239–242, 2006. Cité page 80.

[Ukk02] Esko UKKONEN : Finding Founder Sequences from a Set of Recombinants. *In* Roderic GUIGÓ et Dan GUSFIELD, éditeurs : *Proceedings of the 2nd International Workshop Algorithms in Bioinformatics (WABI)*, volume 2452 de *Lecture Notes in Computer Science*, pages 277–286. Springer, 2002. Cité pages 168, 169, 170, 171 et 172.

[UY00] Takeaki UNO et Mutsunori YAGIURA : Fast Algorithms to Enumerate All Common Intervals of Two Permutations. *Algorithmica*, 26(2) :290–309, 2000. Cité page 178.

[Vaz07] Vijay V. VAZIRANI : *Algorithmes d'approximation*. Springer-Verlag, 2007. Cité pages 54, 57 et 58.

[Via04] Stéphane VIALETTE : On the computational complexity of 2-interval pattern matching problems. *Theoretical Computer Science*, 312(2-3) :223–249, 2004. Cité pages 188 et 189.

[VM00] Gabriele VARANI et William H. MCCLAIN : The G-U wobble base pair. A fundamental building block of RNA structure crucial to RNA function in diverse biological systems. *EMBO Reports*, 1(1) :18–23, 2000. Cité page 65.

[vMKS+02] Christian von MERING, Roland KRAUSE, Berend SNEL, Michael CORNELL, Stephen G. OLIVER, Stanley FIELDS et Peer BORK : Comparative assessment of large-scale data sets of protein-protein interactions. *Nature*, 417(6887) :399–403, May 2002. Cité page 72.

[WG07] Yufeng WU et Dan GUSFIELD : Improved Algorithms for Inferring the Minimum Mosaic of a Set of Recombinants. *In* Bin MA et Kaizhong ZHANG, éditeurs : *Proceedings of the 18th Annual Symposium Combinatorial Pattern Matching (CPM)*, volume 4580 de *Lecture Notes in Computer Science*, pages 150–161. Springer, 2007. Cité pages 168, 169, 170 et 171.

[Wil09] Ryan WILLIAMS : Finding paths of length k in $O^*(2^k)$ time. *Information Processing Letters*, 109(6) :315–318, 2009. Cité page 44.

[Woe01] Gerhard J. WOEGINGER : Exact Algorithms for NP-Hard Problems : A Survey. *In* Michael JÜNGER, Gerhard REINELT et Giovanni RINALDI, éditeurs : *Combinatorial Optimization - Eureka, You Shrink !*, volume 2570 de *Lecture Notes in Computer Science*, pages 185–208. Springer, 2001. Cité page 60.

[Wu10] Yufeng WU : Bounds on the Minimum Mosaic of Population Sequences under Recombination. *In* Amihood AMIR et Laxmi PARIDA, éditeurs :

Proceedings of the 21st Annual Symposium Combinatorial Pattern Matching (CPM), volume 6129 de *Lecture Notes in Computer Science*, pages 152–163. Springer, 2010. Cité page 171.

[YA10] Xiao YANG et Srinivas ALURU : An Improved Model for Gene Cluster Inference. *In* Hisham AL-MUBAID, éditeur : *Proceedings of the ISCA 2nd International Conference on Bioinformatics and Computational Biology (BICoB)*, pages 190–195. ISCA, 2010. Cité page 180.

[YSB+11] Xiao YANG, Florian SIKORA, Guillaume BLIN, Sylvie HAMEL, Romeo RIZZI et Srinivas ALURU : An Algorithmic View on Multi-related-segments : a new unifying model for approximate common interval, 2011. Soumis. Cité pages 15, 177 et 178.

[ZBV09] Mikhail ZASLAVSKIY, Francis R. BACH et Jean-Philippe VERT : Global alignment of protein-protein interaction networks by graph matching methods. *Bioinformatics*, 25(12) :i259–i267, 2009. Cité page 81.

[Zuc07] David ZUCKERMAN : Linear Degree Extractors and the Inapproximability of Max Clique and Chromatic Number. *Theory of Computing*, 3(1) :103–128, 2007. Cité page 122.

Aspects algorithmiques de la comparaison d'éléments biologiques

Florian Sikora

Résumé :

Pour mieux saisir les liens complexes entre génotype et phénotype, une méthode utilisée consiste à étudier les relations entre différents éléments biologiques (entre les protéines, entre les métabolites...). Celles-ci forment ce qui est appelé un *réseau biologique*, que l'on représente algorithmiquement par un graphe. Nous nous intéressons principalement dans cette thèse au problème de la recherche d'un motif (multi-ensemble de couleurs) dans un graphe coloré, représentant un réseau biologique. De tels motifs correspondent généralement à un ensemble d'éléments conservés au cours de l'évolution et participant à une même fonction biologique. Nous continuons l'étude algorithmique de ce problème et de ses variantes (qui admettent plus de souplesse biologique), en distinguant les instances difficiles algorithmiquement et en étudiant différentes possibilités pour contourner cette difficulté (complexité paramétrée, réduction d'instance, approximation...). Nous proposons également un greffon intégré au logiciel Cytoscape pour résoudre efficacement ce problème, que nous testons sur des données réelles.

Nous nous intéressons également à différents problèmes de génomique comparative. La démarche scientifique adoptée reste la même : depuis une formalisation d'un problème biologique, déterminer ses instances difficiles algorithmiquement et proposer des solutions pour contourner cette difficulté (ou prouver que de telles solutions sont impossibles à trouver sous des hypothèses fortes).

Mots clés : bio-informatique, algorithmique, complexité paramétrée, approximation, réseaux biologiques, génomique comparative.

Abstract :

To investigate the complex links between genotype and phenotype, one can study the relations between different biological entities. It forms a *biological network*, represented by a graph. In this thesis, we are interested in the occurrence of a motif (a multi-set of colors) in a vertex-colored graph, representing a biological network. Such motifs usually correspond to a set of elements realizing a same function, and which may have been evolutionarily preserved. We follow the algorithmic study of this problem, by establishing hard instances and studying possibilities to cope with the hardness (parameterized complexity, preprocessing, approximation...). We also develop a plugin for Cytoscape, in order to solve efficiently this problem and to test it on real data.

We are also interested in different problems related to comparative genomics. The scientific method is the same : studying problems arising from biology, specifying the hard instances and giving solutions to cope with the hardness (or proving such solutions are unlikely).

Keywords : bio-informatic, algorithmics, parameterized complexity, approximation, biological networks, comparative genomics.